

# Scalable and Secure Aggregation in Distributed Networks

Sébastien Gambs<sup>1</sup>, Rachid Guerraoui<sup>2</sup>, Hamza Harkous<sup>2</sup>

Florian Huc<sup>2</sup>, and Anne-Marie Kermarrec<sup>3</sup>

<sup>1</sup>Université de Rennes 1 – INRIA/IRISA

<sup>2</sup>École Polytechnique Fédérale de Lausanne (EPFL)

<sup>3</sup>INRIA Rennes Bretagne-Atlantique

**Abstract.** We consider the problem of computing an aggregation function in a *secure* and *scalable* way. Whereas previous distributed solutions with similar security guarantees have a communication cost of  $O(n^3)$ , we present a distributed protocol that requires 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  $(\frac{1}{2} - \epsilon)n$  malicious nodes for any constant  $\frac{1}{2} > \epsilon > 0$  (not depending on  $n$ ), and outputs the exact value of the aggregated function with high probability.

## 1 Introduction

*Aggregation functions* are a specific class of functions that compute a global function from the local values held by nodes in a distributed system. Examples of aggregation functions include simple functions such as the average computation but also more sophisticated ones such as an election with multiple candidates. Such functions are particularly important in large-scale systems in which they are typically used to compute global system properties (e.g. for monitoring purposes). While such computations may be achieved through a trusted central entity gathering all inputs [BFP<sup>+</sup>01], distributed variants are appealing for scalability and privacy reasons.

In this paper, we address simultaneously three issues related to computing an aggregation function in a distributed fashion, namely correctness, privacy and scalability. Indeed, the problem of computing an aggregation function is a specific instance of the much broader problem of secure multiparty computation. Generic constructions can thus be used [GMW87,BOG88] for solving the problem while tolerating up to  $n/2 - 1$  malicious nodes, where  $n$  is the number of nodes in the network. However, these constructions are often expensive with a global communication cost that is quadratic or cubic in the number of nodes in the network [BTH08]. In addition, most of them assume the existence of a broadcast channel, which is rarely available in large scale networks. Simulating such a channel is possible but has a communication cost of  $\Omega(n^2)$ .

Ideally, we would like the communication complexity to compute an aggregation function to be linear in the number of nodes. The fundamental question is whether an algorithm for computing an aggregation function with a communication cost of  $O(n)$  in a secure and accurate way in the presence of a constant fraction of malicious nodes exists. Clearly, this is impossible to achieve with certainty. Indeed, to be certain that a message sent by a node is not altered by a collusion of malicious nodes, it needs to be sent at least  $m + 1$  times, where  $m$  is the number of malicious nodes (otherwise the malicious nodes could simply drop the message). Such an algorithm would induce  $O(nm) = O(n^2)$  messages when  $m = O(n)$ . Therefore, instead of seeking certainty, we investigate algorithms that output the exact value of the aggregation function *with high probability (whp)*. In fact, our first contribution is to prove a lower bound stating that at least  $\Omega(n \log n)$  messages are required to compute any multiparty function (not only aggregation) in an accurate way and with high probability. The lower bound proof leverages a strategy that consists for the adversary to control all the nodes to which a specific node sends its messages, whilst honest nodes hide as much as possible the selection of nodes that will treat their votes.

Our second contribution is a near-optimal distributed protocol for computing the output of any aggregation function while protecting the privacy of individual inputs, and tolerating up to  $(\frac{1}{2} - \epsilon)n$  malicious nodes, for  $n$  the number of nodes in the network and  $\epsilon > 0$  any constant independent of  $n$ . This protocol outputs the exact value of the aggregation function with high probability as  $n$  tends to infinity, has a global communication cost of  $O(n \log^3 n)$ , and is balanced. It is based on a distributed version of the protocol presented in [AS09] to build an overlay of clusters of size  $O(\log n)$ , such that each of them contains a majority of honest nodes (the construction of such a layout is a necessary condition to ensure the accuracy of our protocol.). It uses this layout to compute the aggregation function using existing cryptographic tools at the level of the clusters, that would otherwise be too expensive (in terms of communication complexity) to run at the level of the network.

We support our theoretical evaluations with an implementation of a distributed polling protocol, based on our aggregation protocol, on the Emulab testbed [WLS<sup>+</sup>02]. We show its communication, computation, and time efficiency compared to a non-layout based secure aggregation protocol.

The rest of the paper is organized as follows. In Section 2, we discuss the model and related work. Afterwards, we prove a lower bound on the minimal communication complexity of computing an aggregation function in Section 3. The protocol is described and analyzed in Section 4. Finally, we report on the results of an experimental evaluation of the protocol in Section 5 and we conclude in Section 6.

## 2 Preliminaries and Background

### 2.1 Model

We consider a dynamic and distributed system composed of nodes that can join and leave at any time. We assume that the number of nodes in the system is always linear in a parameter  $n$ . Furthermore, some of the nodes (that we refer to hereafter as *malicious*) might display an undesirable behaviour and cheat during the execution of the protocol in order to learn information of honest nodes, to perturb the functionality of the system, or simply to make it crash. We model these malicious nodes through a static and active adversary controlling and coordinating them (see [Gol01] for a formal definition). At any point in time, the adversary has a complete knowledge of the structure of the system while an honest node has only a local knowledge of its neighbourhood. When a node joins the network, this adversary can corrupt it to make it a malicious node, otherwise the node would be called honest. Of course, during the execution of a protocol, an honest node has no idea whether or not it is currently interacting with an honest node or a malicious node. We assume that  $\tau$ , the fraction of malicious nodes controlled by the adversary, is less than  $\frac{1}{2} - \epsilon$  for some constant  $\frac{1}{2} > \epsilon > 0$  independent of  $n$ . In order to construct the overlay of the system, we rely on a distributed version of the protocol proposed in [AS09]. The analysis of this protocol has been conducted in the following setting: first, only the honest nodes are present in the network and there has been enough leave and join operations from these honest nodes, then the adversary has the malicious nodes join the network. After this initialisation phase, the nodes can leave and join arbitrarily, but still while conserving a maximum fraction  $\tau$  malicious nodes in the system.

*Remark 1.* It has been proved that the protocol of [AS09] guarantees to maintain a partition of the nodes into clusters of size  $\Omega(\log n)$ , such that in each cluster, there is a majority of honest nodes under the hypothesis that the fraction of malicious nodes controlled by the adversary, is less than  $\frac{1}{2} - \epsilon$  and that the adversary cannot force an honest node to leave the system. The partition can be made further resilient to other attacks (particularly denial-of-service attacks

which result in honest nodes leaving the system), by using the protocol proposed in [AS07]. However, in this case, the fraction of malicious nodes controlled by the adversary, is required to be less than  $\frac{1}{5} - \epsilon$ . The results presented in this paper can be straightforwardly adapted to this latter situation.

We further assume that each node has a pair of secret/public keys for signature that is assigned to it by a (trusted) Certification Authority (CA). Note that this CA is dedicated to key management and cannot be used to achieve other computations. Some decentralized implementations of such a CA are possible but they are out of the scope of this paper. More precisely, when a new node joins the network, it receives a certificate signed by the CA that contains its public key and that can be shown to other nodes, as well as the corresponding private key. As the public key of a node is unique and chosen at random, it can be considered as being the identifier of this node. We also assume that all nodes in the system can communicate via pairwise secure channels (these secure channels can be obtained for instance via a public key infrastructure or a key generation protocol), which means that all the communications exchanged between two nodes are authenticated and confidential from the point of view of an external eavesdropper.

Periodically, the nodes currently within the system compute, in a distributed manner, an aggregation function  $f$  that depends on individual inputs of the nodes,  $x_1, \dots, x_n$ , where  $x_i$  is taken from a set of  $k$  different possible values (i.e.,  $x_i \in \{\nu_1, \dots, \nu_k\}$ ). This aggregation function can be used for instance to compute a distributed polling or any global property of the network that can be obtained by a linear combination of the local inputs. In the rest of the paper, we assume that these values correspond to integers. A simple example of such a function is the computation of the average (or the sum) of these values (i.e.,  $f(x_1, \dots, x_n) = \frac{1}{n} \sum_{i=1}^n x_i$ ).

Due to the presence of malicious 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, and this against any actions that the active adversary might do.

**Definition 1 (Privacy – active adversary [Gol01]).** *A distributed protocol is said to be private with respect to an active adversary controlling a fraction  $\tau$  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 nodes he controls) and the output of the protocol.*

Privacy can either be *information-theoretic* (unconditional), if an adversary with unlimited computational power cannot break the privacy of the inputs of honest nodes, or *cryptographic*, if it only holds against a computationally-bounded adversary that does not have enough computing resources to break a cryptographic assumption (such as factorising the product of two big prime numbers or solving the discrete logarithm problem). Furthermore, we aim at ensuring correctness (Definition 2), even if the malicious nodes controlled by the adversary misbehave.

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

However, malicious nodes cannot be denied from choosing their own inputs as they want as long as these inputs are valid (i.e. they are within the range of the possible ones). Neither can we avoid the situation in which a malicious node simply refuses to participate to the protocol, in which case the protocol should be run without taking into account this node. If a malicious node leaves the protocol *during* its execution then the protocol should be robust enough to carry

out the computation instead of simply aborting. This corresponds to a type of denial-of-service attack on the global functionality computed by the protocol.

Moreover, we seek *scalable* protocols whose global communication cost and computational complexity is as close as possible to the lower bound  $\Omega(n \log n)$  that we prove in Section 3. The term *scalable* reflects the property of a protocol to remain efficient when the size of the system increases, and hence its suitability to be implemented in large scale systems. As the notion of scalability imposes a bound on the communication and computational complexity, it has an impact on the type of cryptographic techniques that can be used. For instance as mentioned in Section 2.3, general results in secure multiparty computation can be used to compute any distributed function in a secure manner as long as the number of malicious participants is less than  $n/3$  when aiming for information-theoretic security [CCD88] or less  $n/2$  for achieving cryptographic security [GMW87]. However, these protocols have a communication and computational complexities that are at least quadratic or cubic in the number of nodes  $n$ .

Apart from scalability, 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.*

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 a  $(n, n)$ -balanced protocol a single node may do all the work. In this paper, we will consider as *balanced* a protocol that is  $(Poly(\log n), Poly(\log n))$ -balanced.

## 2.2 Homomorphic encryption

In our work, we rely on a cryptographic primitive known as *homomorphic encryption*, which allows to perform arithmetic operations (such as addition and/or multiplication) on encrypted values, thus protecting the privacy of the inputs of honest nodes.

**Definition 4 (Homomorphic cryptosystem).** *Consider a public-key (asymmetric) cryptosystem where (1)  $\text{Enc}_{pk}(a)$  denotes the encryption of the message  $a$  under the public key  $pk$  and (2)  $\text{Dec}_{sk}(a) = a$  is the decryption of this message with the secret key  $sk$ . (In order to simplify the notation, we drop the indices and write  $\text{Enc}(a)$  instead of  $\text{Enc}_{pk}(a)$  and  $\text{Dec}(a)$  instead of  $\text{Dec}_{sk}(a)$  for the rest of the paper.) A cryptosystem is additively homomorphic if there is an efficient operation  $\oplus$  on two encrypted messages such that  $\text{Dec}(\text{Enc}(a) \oplus \text{Enc}(b)) = a + b$ . Moreover, such an encryption scheme is called affine if there is also an efficient scaling operation  $\odot$  taking as input a cipher-text and a plain-text, such that  $\text{Dec}(\text{Enc}(c) \odot a) = c \times a$ .*

Paillier’s cryptosystem [Pai99] is an instance of a homomorphic encryption scheme that is both additive and affine. Moreover, Paillier’s cryptosystem is also *semantically secure* [Gol01], which means that a computationally-bounded adversary cannot derive non-trivial information about the plain text  $m$  encrypted from the cipher text  $\text{Enc}(m)$  and the public key  $pk$ .

**Definition 5 (Semantic security [Gol01]).** *An encryption scheme is said to be semantically secure if a computationally-bounded adversary cannot derive non-trivial information about the plain text  $m$  encrypted from the cipher text  $\text{Enc}(m)$  and the public key  $pk$ .*

For instance, a computationally-bounded adversary who is given two different cipher texts encrypted with the same key of a semantic cryptosystem cannot even decide with non-negligible

probability if the two cipher texts correspond to the encryption of the same plain text or not. This is because a semantically secure cryptosystem is by essence *probabilistic*, meaning that even if the same message is encrypted twice, the two resulting cipher-texts will be different with high probability. In practice, the semantic security is achieved by injecting in each possible operation of the cryptosystem (encryption, addition and multiplication) some fresh randomness. However, if two participants agree in advance to use the same seed for randomness, this renders the encryption scheme deterministic. For instance, if two different participants each add together two encrypted values,  $\text{Enc}(m_1)$  and  $\text{Enc}(m_2)$ , using the addition operation of the cryptosystem, this normally results in two different cipher-texts (but still corresponding to the same plain-text), *unless* the participants agree in advance to use the same randomness.

In this paper, we use a *threshold version* of the Paillier’s cryptosystem due to D amgaard and Jurik [DJ01].

**Definition 6 (Threshold cryptosystem).** *A  $(t, n)$  threshold cryptosystem is a public cryptosystem where at least  $t > 1$  nodes out of  $n$  need to actively cooperate in order to decrypt an encrypted message. In particular, no collusion of even  $(t - 1)$  nodes can decrypt a cipher text. However, any node may encrypt a value on its own using the public-key  $pk$ . After the threshold cryptosystem has been set up, each node  $i$  gets as a result its own secret key  $sk_i$  (for  $1 \leq i \leq n$ ).*

### 2.3 Related Work – Secure Multiparty Computation

The main goal 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. This problem was first introduced in the bipartite setting in [Yao82] 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 that can be used to compute securely (in the cryptographic sense) any distributed function as long as the number of malicious nodes is strictly less than  $n/2$  [GMW87]. For some functions, it has been proven that these protocols are optimal with respect to the number of malicious nodes that can be tolerated. Subsequently, complementary work [CK91] has shown that the only functions that can be computed even when a majority of nodes are malicious (up to  $n - 1$ ) are those that consist in a XOR-combination of  $n$  Boolean functions. In the information-theoretic setting, generic results for secure multiparty computation also exist that can tolerate up to  $n/3$  malicious nodes without relying on any cryptographic assumptions [BOG88, CCD88]. All these generic constructions have constant overheads (independent of the function to be computed), in terms of communication and computation, that are at least quadratic or cubic in the number of nodes [BTH08]. However, for solving specific tasks it is often possible to develop more efficient protocols. Furthermore, contrary to us, some of these constructions assume the availability of a secure broadcast channel, i.e it is assumed that a broadcast can be performed for a communication cost which is the size of the message. Recently, [HZ10] has proposed a protocol for simulating such a channel that can tolerate up to  $n/2$  malicious nodes with a global communication cost of  $\Theta(n^2)$ . We use this protocol as a subroutine in our approach to broadcast messages to a small set of nodes. Notice that this broadcast protocol is proved to be secure against an adaptive adversary, which is a stronger adversary than the one we consider (the static one). Therefore, one could argue that using this protocol is too strong and that we should rely on a simpler one.

Assuming a particular structure of the underlying network can sometimes lead to efficiency gains. For instance, a recent protocol [BDNP08] achieves the secure multiparty computation of any function in at most 8 rounds of communication. This protocol uses a cluster composed of  $m$  nodes (for  $m < n$ ) to simulate a trusted central entity, has a global communication cost of  $O(m^2 + nm)$  and can tolerate up to  $m/2$  honest-but-curious nodes (a weaker form of

adversary model in which nodes do not cheat actively but can share their data in order to infer new information). This protocol can be used to compute privately an aggregation function as long as the number of honest-but-curious nodes is small (i.e.  $O(\log n)$ ) but does not aim at providing security against malicious nodes as we do. A related technique, frequently used in the context of scalable distributed consensus, is *universe reduction*, which consists in selecting a small subset of the  $n$  nodes, among which the fraction of malicious nodes is similar to the fraction of malicious nodes in the entire system with high probability. This subset of nodes can run expensive protocols and disseminate the results to the rest of the network. Up to our knowledge, no solution for this problem with a sub-quadratic global communication complexity has been found in presence of an adversary controlling a constant fraction of the nodes, without any simplifying assumptions. Feige’s lightest bin protocol [Fei99] solves the problem when all nodes have access to a broadcast channel, and it has been adapted to the message passing model by [KSSV06], with a complexity polynomial in  $n$ , the original network size. Another solution proposed in [BOPV06] has a quasi-polynomial communication complexity (i.e. the total number of bits sent by honest nodes is  $O(n^{\log n})$ ). Other protocols by [KKK<sup>+</sup>08] and [KS10] claim sub-quadratic communication complexity against a malicious adversary controlling less than  $n/(6+\epsilon)$  (for some constant  $\epsilon > 0$ ) and  $n/3$  nodes respectively, but they assume that all nodes know all the identities of all the other nodes in the system, which in itself hides an  $\Omega(n^2)$  communication complexity to propagate this information.

Aggregation functions also have a clear connection with election problems. In particular for these problems, it is of paramount importance to protect the privacy of voters. In [BFP<sup>+</sup>01], a protocol 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. Another protocol [JCJ10] achieves the property of coercion-resistance, i.e. a voter cannot produce any proof of the value of its vote, thus preventing any possibility to force or buy its vote. This protocol has a complexity of  $O(n^2)$ , where  $n$  is the number of voters, which (quoting the authors) is not practical for large scale elections. Finally, our work is to be compared with [GGHK10], in which the authors propose a protocol computing an  $\sqrt{n}$ -approximation of an aggregation function even in the presence  $\sqrt{n}/\log n$  rational nodes (a slightly weaker form of adversary than malicious nodes) with a global communication cost of  $O(n^{3/2})$  but without relying on cryptographic assumptions.

### 3 Lower Bound

In this section, we prove that no balanced algorithm can compute an aggregation function with a global communication complexity  $o(n \log n)$  provided that the number of malicious nodes is linear in the number of nodes in the system.

**Theorem 1.** *Given a protocol that computes a function in a distributed manner, whose inputs are held by  $n$  nodes and among which  $\epsilon n$  are malicious for some positive constant  $\epsilon < 1$  (independent of  $n$ ). If a fraction  $c n$  of the nodes send no more than  $\omega^+(n)$  messages (for some  $\omega^+(n) = o(\log n)$  and constant  $1 > c > 0$ ) and that no node receives more than  $\omega^-(n)$  messages with  $\omega^+(n)e^{\omega^+(n)}\omega^-(n) = o(n)$ , then with high probability (in  $n$ ) there is a node whose messages are all intercepted by malicious nodes.*

*Proof.* If the structure of the overlay is fixed using deterministic arguments before the adversary chooses which nodes he corrupts, he can always target a particular node by corrupting the  $\omega^+(n)$  nodes with whom this honest node will communicate. Similarly, if the set of nodes receiving all the messages of a given node are not chosen uniformly at random, the adversary may possibly use this bias to increase the chances that all the nodes in the determined set are malicious.

The best strategy for a node to avoid to send all its messages to malicious nodes, is to choose the recipients (we call recipient for  $x$ , a node that receives a message from node  $x$ ) of its messages uniformly at random. The objective of our proof is to show that even under this strategy, there is at least a node that will send all its messages to malicious ones. For this, we consider several disjoint sets, each containing all the recipients of a particular node.

To find such disjoint sets of nodes, we proceed in a greedy way. Given a node  $x_1$  that sends less than  $\omega(n)$  messages, we set  $S_1$  to be the set of recipients for  $x_1$ . By assumption, this set intersects at most  $\omega^-(n)\omega^+(n)$  sets of recipients of other nodes. We then proceed recursively by discarding all the nodes sending their messages to these sets. At the end of the process, we obtain  $k = cn/(\omega^-(n)\omega^+(n))$  nodes that send all their messages to disjoint sets  $S_1, \dots, S_k$ .

We now want to prove that with high probability, one of these sets is composed exclusively of malicious nodes. The adversary can choose arbitrarily the nodes he want to corrupt to be malicious. However, we assume that once its choice is made, it is fixed and the adversary cannot change it later. Recall that the sets  $S_i$ ,  $1 \leq i \leq k$  have been chosen at random. Given the set  $S_i$ , we denote by  $S = \cup_{i=1}^k S_i$  and  $m_S$  the number of malicious nodes in  $S$ , the two following processes generate the same probability distribution:

1. Choose  $\epsilon n$  malicious nodes, and then take disjoint sets  $S_1, \dots, S_k$  among the  $n$  possible ones.
2. Draw  $m_S$ , the number of malicious nodes in  $S$ , according to the binomial distribution  $Bi(\epsilon, |S|)$ , and then choose at random  $m_S$  malicious nodes from  $S$ .

Note that *we do NOT assume that the malicious nodes are chosen at random*. Rather we state that if two processes lead to the same probability distribution, proving a result for one of the two processes implies the same result for the other process. Let us recall that the malicious nodes are corrupted by the adversary at the time they join the network.

When choosing  $m_S$  malicious nodes from  $S$ , we have a non-independent probability for a node to be malicious or not, whereas, if we had independence, the proof would be easy. Therefore our objective is now to introduce independence. Notice that choosing  $m_S$  nodes at random in  $S$  gives the same distribution as first choosing  $m'_S < m_S$  nodes at random, and then  $m_S - m'_S$  other nodes at random. Moreover, choosing  $m'_S$  nodes at random has the same probability distribution as the process of choosing malicious nodes such that each node is malicious with probability  $m'_S/S$ , knowing that exactly  $m'_S$  nodes are chosen.

To summarize, the following process leads to the same probability distribution:

1. Choose  $m_S$  according to the binomial distribution  $Bi(\epsilon, |S|)$ .
2. Choose at random and independently for each node if it is malicious with probability  $m_S/2|S|$ . This gives  $m'_S$  malicious nodes.
3. With high probability  $m_S > m'_S$ . Choose  $m_S - m'_S$  other malicious nodes at random.

Step 3 can be proved using standard Chernoff's bound arguments (more precisely by showing that  $m'_S$  is close to  $m_S/2$ ), so if we can prove that after Step 1, there is a set containing exclusively malicious nodes, the theorem is proved. We therefore proceed in this direction.

We denote by  $Mal$  the set containing all the malicious nodes. A set  $S_i$  has the following probability to contains exclusively malicious nodes:  $Prob(|S_i \cap Mal| = |S_i|) \geq (m_S/2|S|)^{\omega^+(n)}$ .  $Q$  is the event in which no set is composed exclusively of malicious nodes and has the following

probability:  $Prob(Q) \leq (1 - (m_S/2|S|)^{\omega^+(n)})^k$ . However, with high probability, we have:

$$\begin{aligned} & \lim_{n \rightarrow \infty} (1 - (m_S/2|S|)^{\omega^+(n)})^k \\ &= \lim_{n \rightarrow \infty} e^{-(m_S/2|S|)^{\omega^+(n)} * k} \\ &= \lim_{n \rightarrow \infty} e^{-cn(m_S/2|S|)^{\omega^+(n)}/(\omega^-(n)\omega^+(n))} \\ &= 0 \end{aligned}$$

because 1)  $\omega^+(n)e^{\omega^+(n)}\omega^-(n) = o(n)$  and 2) with high probability,  $m_S/2|S| > \epsilon/4$ .

Therefore, for big enough  $n$ , we will have an honest node that will send all its messages exclusively to malicious nodes. In this case, the malicious nodes can discard all its messages thus preventing the computation of the output to be exact.

**Corollary 1.** *A  $(Poly(\log n), n)$ -balanced protocol that computes with high probability the exact value of a function in a distributed manner, whose inputs are held by  $n$  nodes among which  $\epsilon n$  are malicious for some positive constant  $\epsilon < 1$  (independent of  $n$ ), induces a total of  $\Omega(n \log n)$  messages.*

*Proof.* By contradiction, suppose that the protocol induces  $o(n \log n)$  messages. We have some  $\omega^+(n) = o(\log n)$  and some constant  $1 > c > 0$  such that  $cn$  nodes send less than  $\omega^+(n)$  messages. As the protocol is  $(Poly(\log n), n)$ -balanced, no node receives more than  $\omega^- = Poly(\log n) * o(\log n) = Poly(\log n)$  messages. These  $\omega^+$  and  $\omega^-$  verify the conditions of Theorem 1, thus implying the existence of an honest node whose messages have been sent only to malicious nodes. Therefore as this node is “surrounded” by malicious nodes, they can decide to discard all its messages, thus preventing its input to be taken into account in the computation, and as a consequence the outcome cannot be exact.

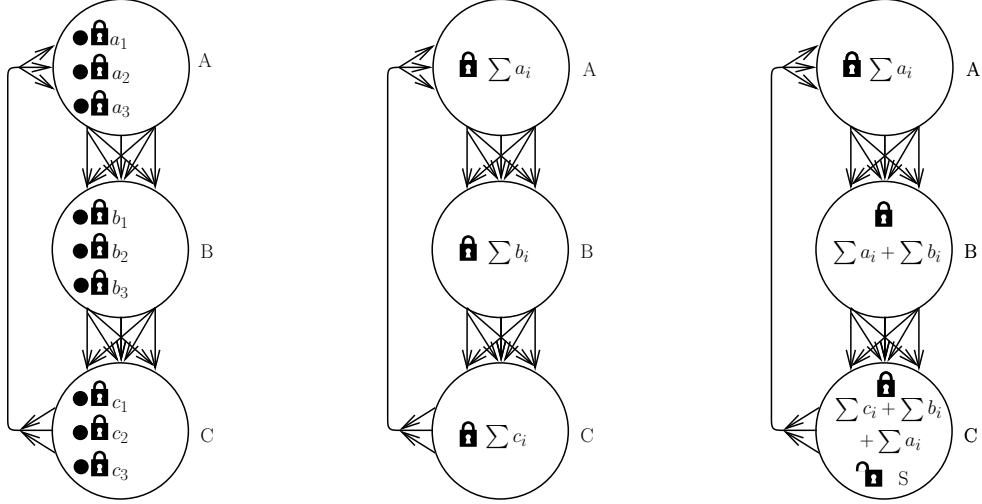
## 4 Distributed Aggregation Protocol

### 4.1 Overview

We now provide a high-level view of the protocol that distributively computes an aggregation function (see also Figure 1). Our protocol uses a distributed version of the protocol of [AS09], which we describe in 4.2. It provides us with a partition of the  $n$  nodes into  $g$  clusters,  $C_1, \dots, C_g$ , of equal size, which are organized in a ring such that all nodes of a cluster  $C_i$  are connected to all nodes of the previous cluster ( $C_{i-1 \bmod g}$ ) and the next cluster ( $C_{i+1 \bmod g}$ ). Assuming that a threshold homomorphic cryptosystem has been set up and that all nodes know the public key  $pk$  of this threshold cryptosystem, the aggregation function can be computed by the following steps:

1. Each node encrypts its input under the public key of the threshold homomorphic cryptosystem and securely broadcasts it to all the other nodes in its cluster. Secure broadcast ensures that (i) all honest players eventually output an identical message, whatever the sender does (consistency), and (ii) this output is the sender’s message, if it is honest (validity) [HZ10].
2. ) The nodes in a given cluster agree on a common random string  $rand$ , and then each node adds the encrypted input it received from its own cluster, using the addition operation of the homomorphic cryptosystem by taking  $rand$  as the randomness injected into the homomorphic encryption. The result of this addition is called the *local aggregate* and is the same for each honest node of the cluster.





**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 ring. After this, the nodes of cluster  $B$  know  $\sum a_i + \sum b_i$ , and those in  $C$ ,  $\sum a_i + \sum b_i + \sum c_i$ . The nodes of the last cluster (here  $C$ ) collaborate to perform the threshold decryption and to obtain  $S$ , the output of the aggregation function.

3. Starting from cluster  $C_1$ , the nodes add their local aggregate with the partial aggregate they received from the previous cluster, with the exception of the nodes of cluster  $C_1$  that have received no partial aggregate and hence skip this phase. The nodes of the cluster then send the result of this aggregation (the new *partial aggregate*) to all the nodes of the next cluster. The nodes of the next cluster then consider as the new partial aggregate the encrypted value appearing in a majority (to correct potentially inconsistent messages that have been sent by malicious nodes). This process is repeated  $g - 1$  times until the partial aggregate reaches the last cluster. Then, we say that the *partial aggregate* has become the *global aggregate*.
4. The nodes of the last cluster perform a threshold decryption of the *global aggregate* revealing the output of the aggregation function.

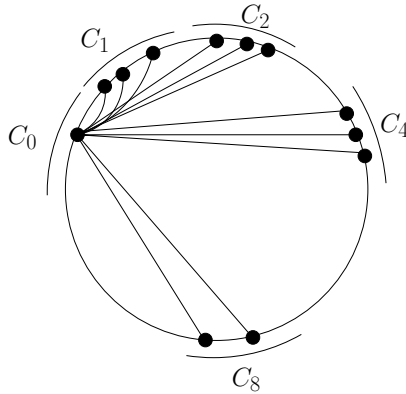
*Remark 2 (Bootstrapping).* The distributed aggregation protocol is meant to be used in a dynamic network, but still a certain stability of the partition of the cluster is required while this computation is taking place. To achieve this, when a node wants to start the computation of an aggregation function, it launches the protocol by broadcasting a time window during which the aggregation protocol will be performed. Any join operation occurring during this time window will be postponed, and similarly, any operation induced by the "leave rule" (cf Section 4.2), except a node leaving itself, will be postponed. Using the layout, any node can perform a broadcast for a communication cost of  $O(n \log n)$  by first broadcasting it to its cluster, and then the clusters recursively forward the message along the ring.

## 4.2 The overlay

For the purpose of the aggregation protocol, we need the nodes to be organized in a layout consisting in a partition of the nodes into clusters  $C_1, \dots, C_g$  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 [AS09]. This protocol uses a central authority and we explain how to discard this requirement at the end of the section, but first we describe its functionality. The central authority

is used to assign to each node a random number between 0 and 1, number which stands for the position of the node in the segment  $[0, 1)$ . By inducing churn when a node joins the system, the protocol ensures that each segment of size  $c \log(n)/n$ , for some specific  $c$ , contains a majority of honest nodes as long as the adversary controls at most a fraction of  $\frac{1}{2} - \epsilon$  of malicious nodes, for some constant  $\frac{1}{2} > \epsilon > 0$ . The clusters  $C_1, \dots, C_g$  are composed of the nodes whose positions are in the respective segments  $[0, c \log n/n), \dots, [1 - c \log n/n, 1)$ .

In order to efficiently distribute the protocol, we first arrange the *clusters* in a Chord-like overlay [YKGGK10]. The adapted join-leave protocol must further ensure that, for all  $1 \leq i \leq g$ , the nodes from cluster  $C_i$  know all the nodes from  $C_{i+2^j}$ , for all  $1 \leq j \leq \log_2 g$ , resulting in  $O(\log^2 n)$  connections per node (c.f. Figure 2).



**Fig. 2.** Chord overlay.

As in [AS09], we assume that initially, the network is composed of only honest nodes (this assumption is also typical in sensor networks or in networks created by trusted peers). This assumption allows to bootstrap the layout construction by building an initial overlay by assigning to each initial node a position in  $[0, 1)$ . We now describe how to adapt the join rule of [AS09], known as the cuckoo rule, and how to maintain the overlay. We further show how to adapt the leave rule of [AS07] that is required when the adversary can force a node to leave the network (for instance through a denial-of-service attack).

We will rely on two subroutines to distribute these rules.

1. Inter-cluster communication: A node from a cluster  $C$  receiving a message from a node of a cluster  $C'$  accepts it if and only if  $C'$  is a neighbor of  $C$  in the Chord overlay, and at least half of the nodes of  $C'$  have sent the same message.
2. Random Number Generation: To generate a number at random (typically between 0 and  $\log n$ ), the nodes of a cluster  $C$  collaborate as follows: each node of  $C$  chooses a random number in the desired range, and commits it to the other nodes of  $C$  via a secure broadcast. Once all nodes have done so or after a given time-out if some nodes do not participate, the committed values are revealed by sending the decryption keys again using the secure broadcast. All the valid random numbers are added modulo some predecided upper bound ( $\log n$  in our example) and the result corresponds to the generated random number, which is agreed upon by all the honest nodes.

We can now explain how the cuckoo rule is modified in order to discard the requirement of a central authority

*Join rule or cuckoo rule:* when a node  $x$  joins the network, we suppose that it can come in contact with one of the node of the network. The node gives the composition of an arbitrary cluster to  $x$  (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 performing the cuckoo rule from [AS07], which corresponds to choosing a position at random in  $[0, 1)$  for  $x$ . Then the nodes of the cluster containing the chosen position, which we call  $C$ , are informed via messages routed using the Chord overlay that  $x$  is inserted at this position. At this point, extra churn is induced, and 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 the previously chosen position (c.f. [AS07] 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 that become adjacent to this node, are informed of the change by messages sent by the nodes of  $C'$  and  $C''$  respectively. This is important since we need that at any time, if two clusters  $C$  and  $C'$  are adjacent in the Chord overlay, all nodes of  $C$  should know the exact composition of  $C'$  to decide whether or not it accepts a message from nodes of  $C'$  during inter-cluster communication.

When the adversary can force any honest node to leave the network, it is shown in [AS07] that churn needs to be introduced whenever a node leaves. For this, they designed a leave rule that can be adapted as follows:

*Leave rule:* when a node at position  $p$  leaves the system, the cluster  $C$  to which it belongs chooses a segment of  $[0, 1)$  of length  $k/n$  (recall that  $k > \frac{1}{\epsilon}$ ). All the nodes of this segment replace the nodes of a randomly chosen segment of same length contained in  $C$ . The old nodes of the replaced segment are moved to position chosen at random in  $[0, 1)$ . Similar to the join operation, all the nodes that were adjacent or that become adjacent to the moved nodes are informed of these changes.

With high probability, the communication overhead of the distributed version of this protocol is  $O(\log^3 n)$  per join or leave operation. Indeed, with high probability, the number of nodes in a segment of size  $k/n$  is  $O(k)$ , therefore with high probability the protocol needs to generate a constant number of random numbers (for a constant depending on  $k$  and therefore on  $\epsilon$ ). This protocol ensures that each cluster contains  $\Omega(\log n)$  nodes, among which there is a majority of honest nodes as long as  $\tau \leq 1/2 - \epsilon$  when one do not consider denial-of-service [AS09], or for  $\tau \leq 1/5 - \epsilon$  otherwise [AS07].

### 4.3 Aggregation computation

In this section, we present the second phase of the protocol that computes the aggregation function. This protocol is optimal up to a logarithmic factor in terms of scalability (i.e. communication and computational complexity) and achieves perfect privacy, correctness against a computationally-bounded adversary controlling at most  $(1/2 - \epsilon)n$  malicious nodes, for a constant  $\frac{1}{2} > \epsilon > 0$  independent of  $n$  and is balanced. For this, we rely on cryptographic techniques that would be too expensive to use on the whole network, but whose costs are efficient when computed at the granularity of a cluster in the structured overlay.

**Setting up the threshold cryptosystem (Step 1)** One particular cluster (that we refer thereafter as the *threshold cluster*) will be in charge of setting up the threshold cryptosystem. For instance, we can decide arbitrarily this threshold cluster to be the last one on the ring. This setup phase requires that all the nodes of this threshold cluster engage themselves in a

distributed key generation protocol [NS11] for the threshold homomorphic cryptosystem [DJ01]. At the end of this key generation phase, all the nodes in the threshold cluster receive the same public key  $pk$  and each one of them gets as private input a different secret key, respectively  $sk_1, \dots, sk_{k \log n}$ . 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, we can set  $t$  to be close to  $\frac{k \log n}{2}$  to ensure that there will be enough honest nodes to cooperate for the final decryption of the result at the end of the protocol. Once the threshold cryptosystem has been set up, the public key  $pk$  is communicated in the network cluster by cluster, following the ring structure. Thereafter, when we say that *a cluster communicates something to the next cluster*, we mean that all the nodes in the current cluster communicate the same value to all the nodes in the next cluster, which results in a communication cost of  $O(\log^2 n)$ . Once, a node in one cluster has received all the  $k \log n$  messages from the previous cluster, it decides to keep as final value for this particular round of intercluster communication the value obtained by performing a majority vote. This value always corresponds to the unique value sent by all the honest nodes of the previous cluster, which are themselves in majority inside this cluster due to the property of the structured overlay.

**Local aggregation (Step 2)** The nodes within a cluster communicate their value encrypted under the public key  $pk$  of the homomorphic encryption system to all the other nodes of the cluster through a secure broadcast channel. When a node sends a message in such a channel, it is received by all the other nodes of the cluster, who know the identity of the sender of this message. The complexity of constructing such a channel is polynomial in the number of nodes of the cluster, which in our case is  $O(\log^2 n)$  and can be obtained for instance by using a recent construction of Hirt and Zikas [HZ10] as long as the number of malicious nodes is less than half of the nodes, which is by assumption the case in our protocol. All the members of the current cluster encrypt their inputs under the public key  $pk$  and broadcast this encrypted value to all the members of the cluster along with a non-interactive zero-knowledge proof that this input is valid [YHM<sup>+</sup>09] (i.e. chosen from the range of valid values  $\{\nu_1, \dots, \nu_k\}$ ). The purpose of this non-interactive zero-knowledge proof is to prevent an adversary from tampering with the output of the protocol by providing as input an arbitrary value that is outside the range of the possible ones. The privacy of the inputs is protected by the semantic property of the cryptosystem. Once all the nodes in the cluster have received the  $\log n$  encrypted inputs from the other members, they add them together using the additive property of the homomorphic cryptosystem. This produces as output the encryption of the sum of all the inputs within this cluster. 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  $rand$ . This value can be obtained for instance by concatenating all the encrypted inputs and then hashing them to obtain a unique random value.

**Global aggregation and threshold decryption (Step 3).** The global protocol proceeds iteratively during  $\frac{n}{\log n}$  iterations, cluster by cluster, starting from the first cluster and following the ring structure of the overlay. At each iteration, the current cluster performs the local aggregation as described above and also aggregates the sum of the local inputs with the encrypted value received from the previous cluster to compute the partial aggregate. This partial aggregate corresponds to the global aggregation of the inputs of the clusters visited so far. As mentioned previously, the encrypted value received from the previous cluster can be computed locally by each node of the current cluster by making a majority vote on the  $k \log n$  messages received from the previous cluster. Once the threshold cluster has been reached (i.e. the last cluster on

the ring), the members of this cluster add their local aggregated values to the encrypted value received from the previous cluster. This produces an encryption of the sum of all the values. Finally, the members of the threshold cluster cooperate to decrypt this encrypted value by using their private keys. 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 [DJ01]. As the number of nodes needed to decrypt successfully is  $t = \frac{k \log n}{2}$  and that we have 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 ring structure of the overlay.

#### 4.4 Analysis of the protocol

During the protocol, due to the use of the Paillier's cryptosystem all the messages exchanges will be of constant size. The value of the constant is directly proportional to a security parameter of the cryptosystem and will not be larger than  $\log n$ . In the following, we count it as  $O(\log n)$ .

**Lemma 1 (Communication cost of the protocol).** *The protocol for the distributed computation of an aggregation function has an overall communication complexity of  $O(n \log^3 n)$  and 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 for an average of  $O(\log^3 n)$  bits of information per node.*

*Proof.* During the setup of the threshold cryptosystem (Step 1) and the threshold decryption (Step 3), 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 (Step 2), 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 intercluster communication requires that all the  $n$  nodes send  $O(\log n)$  messages, each of size  $O(\log n)$ , to the nodes of the next cluster, resulting in a communication cost of  $O(n \log^2 n)$ . As a result, the protocol is dominated by Step 2 (the local aggregation), 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.

**Lemma 2 (Security of the protocol).** *The protocol for the distributed computation of an aggregation function ensures perfect security against a computationally-bounded adversary, tolerates  $(\frac{1}{2} - \epsilon)n$  malicious nodes for any constant  $\frac{1}{2} > \epsilon > 0$  (not depending on  $n$ ), and outputs the exact value of the aggregated function 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 malicious 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 majority vote performed 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 malicious nodes from altering the output during the last step of the protocol.

## 5 Experimental Evaluation

In this section, we evaluate the practical performance of the protocol through experimentations. More precisely, we compare the communication and time complexity of our protocol to a non-layout based protocol. We implemented a distributed polling protocol with two choices to select from, which is an instance of an aggregation protocol, where the nodes’ votes are encoded into inputs for the aggregation protocol and decoded at the end. The experiments were run on Emulab [WLS<sup>+</sup>02], a distributed testbed allowing the user to choose a specific network topology using NS2 configuration file. In each experiment, we used up to 80 *pc3000* machines, Dell PowerEdge 2850s systems, with a single 3 GHz Xeon processor, 2 GB of RAM, and 4 available network interfaces. Each machine ran Fedora 8 and hosted 10 nodes at a time. The nodes were connected to the router in a star topology, setting the maximum network bandwidth to 1000Mb, and the communication relied on UDP. We used the “Authenticated Double-Echo Broadcast algorithm” (as described in [CGR11]) for secure broadcast and the “Paillier Threshold Encryption Toolbox” [DL11] for threshold encryption. The nodes adjust their message sending rate to be uniformly distributed between 0 and 2 seconds.

The only protocol that comes close to our protocol in terms of guarantees is a non-layout based protocol in which each node 1) securely broadcasts its encrypted input to all the 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 output. Such a protocol provides privacy and correctness with certainty against an honest majority for a communication complexity of this protocol is  $O(n^3)$ . Therefore, even for medium sized networks under test (200-800 nodes), it reaches prohibitive complexities, in the order of hundreds of millions of messages exchanged. Accordingly, to evaluate this protocol in comparison to ours, we measured the complexity of single secure broadcast instances. Since the broadcast instances are assumed to be run in sequence to avoid congestion, we can simply add the communication and time complexities. For ease of implementation, we used a centralized key generation authority for setting up the threshold Paillier cryptosystem, whose modulus is fixed to 1024 bits. When evaluating our protocol, we included the overhead of generating keys for the threshold cryptosystem, whereas we assume that the threshold cryptosystem is already set up for non-layout based protocol. This is justified by the fact that these keys can be kept longer than in our case, where the keys need to be generated each time the layout is changed. Moreover, we take the cluster size to be  $20 * \log n$ , which was found to be adequate for an adversary fraction of  $\frac{3}{10}$ .

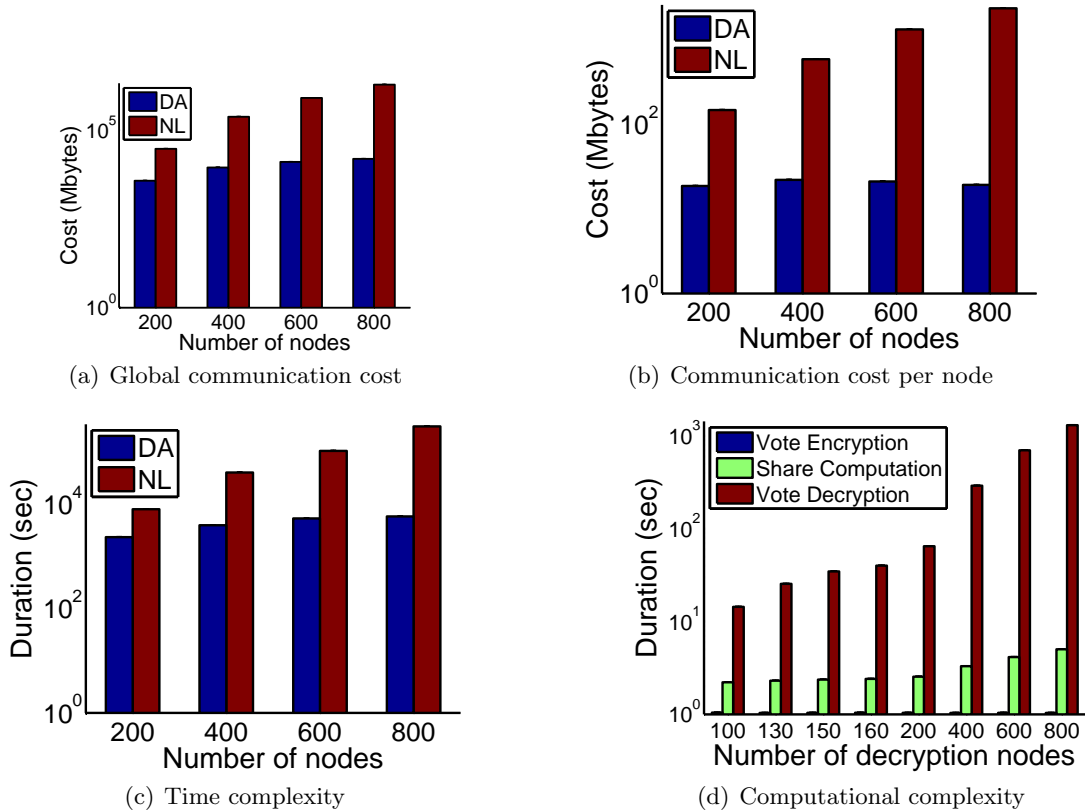
### a) Communication Cost:

Communication cost is the total number of Megabytes sent during the protocol execution. Figure 3.a depicts the global communication cost for the two protocols, on a semi-log scale, with a varying network size. It is apparent that our protocol (labelled as *DA*) features a much smaller cost growth than the non-layout based one (labelled as *NL*) as the number of nodes increases. With only 200 nodes in the network, the *NL* protocol is expected to require around 30 GB overall! To get further insight, we plot in Figure 3.b the communication cost per node. It is noticeable that the growth now is negligible with *DA* protocol, with a 20 MB cost per node while the trend remains as before with *NL*, reaching 600 MB per node cost in a 400 nodes network.

b) *Execution Duration*: In this experiment, for each network size, we average the time taken by a complete execution of the two protocols over all the nodes. This is represented

by the bars in Figure 3.c. Similar to the previous experiment, the growth is much smoother with our *DA* protocol. While the *NL* protocol reaches tens of hours of execution duration, our protocol remains within the scale of minutes. The major part of this duration is communication delays as apparent from Figure 3.d, which shows the duration of the major computationally expensive steps. Particularly, we plot the durations of the vote encryption, share computation, and vote decryption (share combination) for the different decryption network sizes. The first four correspond to the decryption cluster size in our protocol consecutively mapped to the same network sizes as before (200, 400, 600, 800). This figure highlights the efficiency of our protocol and shows that computational complexity is small when compared to the global time complexity. It further indicates that decryption is the most expensive step in such protocols, thus supporting our technique in delegating this task to small clusters.

It is important to note that the goal of this implementation was to provide the comparison between the two protocols and not to provide accurate performance measures of our protocol under various scenarios. Otherwise, several optimizations on the message size and message complexities can be integrated to reduce the complexity in real-world implementations.



**Fig. 3.** Experimental evaluation of our protocol (DA) against a non-layout based one (NL).

## 6 Conclusion

In this paper, we have proposed a distributed protocol that computes aggregation functions in a distributed, scalable and secure way. The complexity of our protocol is drastically lower than those of previously known algorithms and within a factor  $\log^2 n$  of the optimal. Note that using

a binary tree structure instead of a ring allows to reduce the number of communication rounds from  $\frac{n}{\log n}$  currently to  $\log^2 n$ . This can be obtained by a slight modification of the protocol for constructing the structured overlay but would not change the overall communication of the protocol. We leave as open the possibility of deriving a protocol closing the gap between the lower bound of  $\Omega(n \log n)$  and the upper bound  $O(n \log^3 n)$  achieved by our algorithm. We also leave as future work the possibility of adapting our protocol to achieve other multiparty computations than aggregation functions and the possibility of tolerating strictly less than  $< n/2$  instead of  $(1/2 - \epsilon)n$ .

## References

- [AS07] Baruch Awerbuch and Christian Scheideler. Towards scalable and robust overlay networks. In *Proc. 6th Int. Workshop on Peer-To-Peer Systems (IPTPS)*. Citeseer, 2007.
- [AS09] Baruch Awerbuch and Christian Scheideler. Towards a scalable and robust DHT. *Theory of Computing Systems*, 45(2):234–260, February 2009.
- [BDNP08] A. Ben-David, N. Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 257–266. ACM, 2008.
- [BFP<sup>+</sup>01] O. Baudron, P.A. Fouque, D. Pointcheval, J. Stern, and G. Poupard. Practical multi-candidate election system. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 274–283. ACM, 2001.
- [BOG88] M Ben-Or and S Goldwasser. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *on Theory of computing*, 1988.
- [BOPV06] Michael Ben-Or, Elan Pavlov, and Vinod Vaikuntanathan. Byzantine agreement in the full-information model in  $o(\log n)$  rounds. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, STOC '06, pages 179–186, New York, NY, USA, 2006. ACM.
- [BTH08] Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. In *Proceedings of the 5th conference on Theory of cryptography*, pages 213–230. Springer-Verlag, 2008.
- [CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. *Proceedings of the twentieth annual ACM symposium on Theory of computing - STOC '88*, pages 11–19, 1988.
- [CGR11] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.
- [CK91] Benny Chor and Eyal Kushilevitz. A Zero-One Law for Boolean Privacy. *SIAM Journal on Discrete Mathematics*, 4(1):36, 1991.
- [DJ01] I Damgård and M Jurik. A Generalisation, a Simplification and Some Applications of Paillier’s Probabilistic Public-Key System. In *Public Key Cryptography*, number December, pages 119–136. Springer, 2001.
- [DL11] Data and Privacy Lab. Paillier threshold encryption toolbox, July 2011.
- [Fei99] Uriel Feige. Noncryptographic selection protocols. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 142–, Washington, DC, USA, 1999. IEEE Computer Society.
- [GGHK10] A Giurgiu, R Guerraoui, K Huguenin, and A-M Kermarrec. Computing in Social Networks. *Lecture Notes in Computer Science*, 6366/2010:332–346, 2010.
- [GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.
- [Gol01] Oded Goldreich. *Foundations of cryptography: Basic tools*. Cambridge Univ Press, 2001.
- [HZ10] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. *Advances in CryptologyEUROCRYPT 2010*, pages 466–485, 2010.
- [JCJ10] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. *Towards Trustworthy Elections*, pages 37–63, 2010.
- [KKK<sup>+</sup>08] Bruce Kapron, David Kempe, Valerie King, Jared Saia, and Vishal 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, pages 1038–1047, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.
- [KS10] Valerie King and Jared Saia. Breaking the  $o(n^2)$  bit barrier: Scalable byzantine agreement with an adaptive adversary. In *Proceeding of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, PODC '10, pages 420–429, New York, NY, USA, 2010. ACM.



- [KSSV06] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, SODA '06, pages 990–999, New York, NY, USA, 2006. ACM.
- [NS11] Takashi Nishide and Kouichi Sakurai. Distributed paillier cryptosystem without trusted dealer. *Information Security Applications*, pages 44–60, 2011.
- [Pai99] P Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Advances in Cryptology EUROCRYPT'99*, 1999.
- [WLS<sup>+</sup>02] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.
- [Yao82] AC Yao. Protocols for secure computations. *Proceedings of the 23rd Annual IEEE Symposium on*, pages 160–164, November 1982.
- [YHM<sup>+</sup>09] T. Yuen, Qiong Huang, Yi Mu, Willy Susilo, D. Wong, and G. Yang. Efficient non-interactive range proof. *Computing and Combinatorics*, pages 138–147, 2009.
- [YK GK10] Maxwell Young, A. Kate, Ian Goldberg, and M. Karsten. Practical robust communication in DHTs tolerating a byzantine adversary. In *2010 International Conference on Distributed Computing Systems*, pages 263–272. IEEE, 2010.