# Secure and Trustable Distributed Aggregation based on Kademlia

Stéphane Grumbach, Robert Riemann

# Secure and Trustable Distributed Aggregation based on Kademlia

Stéphane Grumbach and Robert Riemann

Inria Grenoble Rhône-Alpes
{stephane.grumbach,robert.riemann}@inria.fr

**Abstract** Aggregation of values that need to be kept confidential while guaranteeing the robustness of the process and the correctness of the result is required in an increasing number of applications. We propose an aggregation algorithm, which supports a large spectrum of potential applications including complex voting protocols. It relies on the distributed hash table Kademlia, used in BitTorrent, for pseudonymous communication between randomly predetermined peers to ensure a high degree of confidentiality which does not solely relies on cryptography. The distribution of data and computation limits the potential for data breaches, and reduces the need for institutional trust. Experimental results confirm the complexity of $\mathcal{O}(\log n)$ for $n$ peers allowing for large-scale applications.

**Keywords:** distributed aggregation, DHT, privacy, trust

## 1 Introduction

An increasing number of applications require aggregation of values that should not be revealed, for various aspects of privacy protection. They include personalized services related to domotic, smart cities, or mobility for instance that are blooming today, while revealing security breaches. Confidentiality protecting aggregation is of even greater importance for online voting. We demonstrate that peer-to-peer systems offer great promises for such aggregations, because they limit the potential for data breaches and simplify the essential question of trust.

This paper presents Advokat, a distributed protocol for confidential aggregation of inputs produced by large sets of peers. It relies on the distributed hash table Kademlia [1], that offers both an overlay network to organize peers, as well as a tree structure to compute the aggregation. Kademlia is a robust and scalable technique which is used in particular by BitTorrent [2]. The proposed protocol integrates also techniques from Bitcoin [3] and BitBallot [4].

Voting is the main privacy preserving aggregation realized with pre-digital technologies. Paper-based voting protocols offer an unmatched solution to satisfy often contradicting though essential properties, such as secrecy of the ballot, correctness of the tally and verifiability. Moreover, the possibility given to voters to participate in the supervision on-site of both the casting and tallying procedures ensures trust. No expert knowledge is required to understand the protocol and

its verification procedure. Thus, no trust in organizing authorities is necessary. Paper-based voting owes its robustness to its independence from institutional trust. Our objective is to transfer as much as possible these properties in the online world, while offering new properties not available in the classical setting, such as remote participation as well as the capacity to launch a new aggregation.

The case of voting protocols is particularly interesting due to its conflicting, but essential security requirements. On one hand, the eligibility of every voter to cast a ballot must be ensured, while, on the other hand, no link can be established between a given ballot and the corresponding voter. Furthermore, the final tally must be verifiable. Distributed protocols are promising for voting since they allow to reduce the reliance on trust and open new prospects for verification. The various tasks are carried out collectively in a peer-to-peer manner by the participants, much like voters in paper-based voting.

We assume the existence of an administrator trusted to certify the eligibility of peers. Supported by a tracker, eligible peers join a Kademlia DHT that provides a tree-like overlay network in which peers are assigned to random leaves. Peers pull inputs, and later input aggregates, from other peers close to them in the tree overlay, which allows to compute aggregates for all ancestor nodes up to the root. The strategy resides on pulling versus pushing for dissemination. Cryptographic signatures are used to authorize peers to pull in different subtrees.

Although several protocols propose a distributed aggregation over tree-like overlay networks [5, 6, 7, 8], to the best of our knowledge, the proposed algorithm is the first to consider eligibility, confidentiality, scalability and verifiability at once. The DASIS protocol [9] balances the Kademlia tree by routing joining peers to less populated subtrees. Internally, the subtree size is computed in a similar fashion to our approach, but no security measures are introduced. A distributed, Kademlia-based voting protocol to rank the quality of BitTorrent content has been proposed [10]. However, confidentiality and eligibility are not addressed.

Using distributed protocols for voting is a very natural idea to avoid concentration of power. Common building blocks, like blind signature schemes [11], Mix Networks [12] or threshold decryption [13] exercise decentralization on a small scale. Many classical online voting protocols employ already a set of authorities [14, 15, 16] to achieve privacy. However, they assume trust in the authorities and the aggregation is generally centralized, rendering the protocols vulnerable to DDoS attacks and data breaches of global impact for instance.

Various efforts[1] are ongoing to propose distributed online voting protocols, based on the Bitcoin blockchain [3], that does not require trusted authorities. Still, published results are sparse. [17] describes a protocol for a binary majority voting to determine the receiver of a voter sponsored Bitcoin payment.

The *SPP* protocol [18], based on Secure Multi-Party Computation, partitions the aggregation over a tree hierarchy of peers of which a random set of peers serves as authorities to carry out the final decryption step. *DPol* [19] and its extension

[1] Blockchain-based voting techniques include: http://votem.com, http://cryptovoter.com, http://votosocial.github.io, http://followmyvote.com, http://bitcongress.org, http://github.com/domschiener/publicvotes

*EPol* [20] are similar to our protocol in that the aggregation is distributed to all peers and for their renunciation of cryptography. However, their message complexity does not allow for large-scale elections.

The aggregation protocol is evaluated with respect to the security properties used for centralized voting protocols such as FOO [15], and to scalability properties used for distributed aggregation protocols such as SPP [19]. We consider eligibility, confidentiality (secrecy), completeness and correctness, verifiability, and complexity in terms of messages, memory and time.

The paper is organized as follows. In the next section, we present the general setting of the protocol. The basic aggregation is shown in Section 3, while the recursive process that takes advantage of the tree overlay of the Kademlia DHT is shown in Section 4. Then, in Section 5, the recursive aggregation process is extended to allow a minority of dishonest Byzantine peers. Several desirable security and complexity properties are sketched in Section 6. The provided confidentiality is experimentally examined in Section 7 by means of a simulation.

## 2 Aggregation Protocol

The protocol relies on *peers*, an *administrator* and a *tracker*. The administrator is entrusted to certify the eligibility of peers. For this purpose, we assume an authenticated, tamper-resistant communication channel between the administrator and each peer, e.g. using an existing public key infrastructure.

Once certified, *peers* join a distributed hash table (DHT) that is mainly used to find other peers, but allows also to retrieve and store data. We choose the Kademlia DHT [1] whose tree-like network overlay is well-suited for aggregations. Like in BitTorrent, a *tracker* is employed to provide an initial peer as an entry point. Peers communicate via pairwise channels assumed to be confidential and authenticated to the degree of a peer pseudonym, e.g. a public IP address.

We use the following notations adapted from [15]:

| | |
|---|---|
| $A$ | Administrator |
| $P_i$ | Peer, $i$-th out of $n$ |
| $a_i$ | Initial aggregate of peer $P_i$ |
| $(pk_i, sk_i)$ | public and private key pair of peer $P_i$ |
| $\eta(m)$ | Hashing technique for message $m$, e.g. SHA-1 |
| $\sigma_i(m)$ | Peer $P_i$'s signature scheme using $(pk_i, sk_i)$ |
| $\sigma_A(m)$ | Administrator's signature scheme |
| $\chi(m, r)$ | Blinding technique for message $m$ and random number $r$ |
| $\delta(s, r)$ | Retrieving technique of blind signature |

The proposed protocol follows the following structural steps:

**Preparation** Peers create personal public and private key pairs and send authorization requests with their blinded public key to the administrator.

**Administration** Once for each peer, the administrator signs the peer's blinded public key without learning it and sends the signature to the peer.

**Aggregation** Supported by the tracker, peers join the tree-like overlay network of Kademlia. Then, peers assign their *initial aggregate* to their leaf node and compute collectively the *root aggregate* from all initial aggregates using the *distributed aggregation algorithm*. This requires the computation of *intermediate aggregates* for all their ancestor nodes in the Kademlia tree.

**Evaluation** On fulfilment of a well-defined verification criteria, peers accept their root aggregate as *final root aggregate*. The outcome (e.g. election result) is eventually derived from the final root aggregate.

In the preparation step, each peer $P_i$ generates on it's own authority a public and private key pair $(pk_i, sk_i)$ to sign messages with $\sigma_i(m)$. To limit the number of valid keys to one per eligible peer, the public key must have the signature of the administrator $A$ [18]. As in FOO [15], a blind signature scheme [11] is used to ensure that $A$ cannot recognize peers after the administration step. $P_i$ randomly chooses a blinding factor $r_i$, computes its blinded public key $b_i = \chi(pk_i, r_i)$ and sends it to $A$ using the authenticated, tamper-resistant channel.

In the administration step, $A$ ensures to sign only a unique $b_i$ for each $P_i$ and responds to $P_i$ with its signature $s_i = \sigma_A(b_i)$. Eventually, $P_i$ can retrieve the *authorization token* $t_i = \delta(s_i, r_i)$. $A$ does not intervene any further once all eligible peers have acquired their authorization or a time-out has elapsed.

During the aggregation step, all peers run the distributed aggregation algorithm, that is presented hereafter in Section 3 and 4.

## 3 Basic Aggregation

The aggregation algorithm allows to implement various kinds of confidential aggregations. In particular, with standard security requirements slightly weakened, it supports a large spectrum of voting systems.

*Aggregates* are values to be aggregated, whether initial aggregates, constituting inputs from peers, or intermediate aggregates obtained during the computation. The specification of the aggregation algebra is formulated below. We then introduce the *aggregate container* allowing to attach meta-information to aggregates that is used to position them in the tree and ensure verifiability.

We introduce an algebra whose operation applies to *aggregates*, which are aggregated during the computation of the operation. In the case of a vote, aggregates correspond to ballot boxes filled with ballots, and the operation is the union of sets. The data structure can be adapted to different applications with different aggregation functions, such as average, majority voting, etc.

We consider a set $\mathbb{A}$ of aggregates. The aggregation operation, $\oplus$, combines two *child aggregates* to a *parent aggregate* in $\mathbb{A}$. *Initial aggregates*, corresponding to peer inputs, are not computed, but provided by the peers. We assume that the operation $\oplus : \mathbb{A} \times \mathbb{A} \mapsto \mathbb{A}$ is commutative and associative.

Consider for illustrative purposes the algebra for the *Plurality Voting* (PV). Peers, or here more precisely voters, choose one out of $d$ options, that are

modeled in the algebra with initial aggregate vectors $(e_1, \ldots, e_d)$ in $\mathbb{A} = \mathbb{N}^d$, with $\sum_{x=1}^{d} e_x = 1$. The operation $\oplus$ is simply vector addition in $\mathbb{A}$. The root aggregate $a_R = (n_1, \ldots, n_d)$ with $\sum_{x=1}^{d} n_x = n$ indicates how many peers $n_x$ have chosen each option. The option $x$ with the highest $n_x$, hence plurality, corresponds to the vote outcome. The system can be easily extended to $\mathbb{A} = \mathbb{Q}_+^d$ to support vote splitting between two or more options. The Manhattan norm is used to ensure the validity of initial aggregates $a_i$ with constant weight: $\|a_i\|$.

More complex voting systems such as for instance the *Alternative Voting* and the *Single Transferable Voting* systems can easily be encoded. In both cases, voters have to rank options. Every ranking of $d!$ possible rankings in total can be interpreted as one option in the PV algebra. The set of aggregates $\mathbb{A}$ consists of vectors $\mathbb{A} = \mathbb{N}_0^{d!}$ and the operation is again vector addition. Note that alternative, more compact encodings can be defined for efficiency reasons.

The aggregation algorithm relies on meta-information of an aggregate $a$ that is in general not directly involved in the aggregate computation, and constitutes together with $a$ the aggregate container of $a$:

| | |
|---|---|
| $h$ | hash $\eta(\cdot)$ of the aggregate container without $h$ |
| $a$ | aggregate |
| $c$ | counter of initial aggregates in $a$, $c = c_1 + c_2$ |
| $c_1$, $c_2$ | counter of initial aggregates of child aggregates |
| $h_1$, $h_2$ | container hashes of child aggregates |
| $\widehat{\mathbb{S}}(x, d)$ | identifier of subtree whose initial aggregates are aggregated in $a$ |

The counter $c$ allows to detect protocol deviations and to measure the number of initial aggregates in the root aggregate that can be compared to $n$ [6].

The aggregate container hash $h$ depends on its child aggregate hashes $h_1, h_2$. As such, a chain of signatures is spanned reaching from the root or any intermediate aggregate down to the initial aggregates of the peers. Also employed in the Bitcoin blockchain, this technique ensures that the sequence of aggregate containers is immutable.

## 4 Recursive Aggregation over the Kademlia Binary Tree

The aggregation protocol relies on the Kademlia DHT that establishes a binary tree overlay network in which each peer $P_i$ is assigned to a leaf node. Using the aggregation operator $\oplus$, peers compute the intermediate aggregate for all the parent nodes from their corresponding leaf up to the root node of the tree. The aggregates used to compute any intermediate aggregate of a given tree node are those of its child nodes. Hence, aggregates have to be exchanged between peers of *sibling subtrees*, i.e. subtrees whose roots have the same parent. Kademlia is not used solely to discover other peers, but its internal tree overlay also provides the hierarchy for the aggregation algorithm [9]. We use in the following a notation adapted from Kademlia [1].

| | |
|---|---|
| $k$ | maximum number of contacts per Kademlia segment ($k$-bucket) |
| $x$ | a Kademlia leaf node ID (KID) of size $B$ |
| $B$ | size of a KID in bits, e.g. 160 |
| $x_i$ | KID of peer $P_i$ |
| $d$ | node depth, i.e. number of edges from the node to the tree root |
| $\widehat{\mathbb{S}}(x,d)$ | subtree whose root is at depth $d$ which contains leaf node $x$ |
| $\mathbb{S}(x,d)$ | *sibling subtree* whose root is the sibling node of the root of $\widehat{\mathbb{S}}(x,d)$ |

The leaf node identifiers $x \in \{0,1\}^B$ ($B$ bits) span the Kademlia binary tree of height $B$ and are denoted KID. Each peer $P_i$ joins the Kademlia overlay network using its KID defined as $x_i = \eta(t_i)$ with the authorization token $t_i$ and the hashing technique $\eta$. This way, $x_i$ depends on both $P_i$'s and $A$'s key pair, so that $x_i$ cannot be altered unilaterally [21]. $B$ is chosen sufficiently large, so that hash collisions leading to identical KIDs for distinct peers are very unlikely. Consequently, the occupation of the binary tree is very sparse.

Any node in the tree can be identified by its depth $d \in \{0,\ldots,B\}$ and any of its descendant leaf nodes with KID $x$. A *subtree* $\widehat{\mathbb{S}}(x,d)$ is identified by the depth $d$ of its root node and any of its leaf nodes $x$. We overload the subtree notation to designate as well the set of players assigned to leaves of the corresponding subtree. Further, we introduce $\mathbb{S}(x,d)$ for the sibling subtree of $\widehat{\mathbb{S}}(x,d)$, so that $\widehat{\mathbb{S}}(x,d) = \widehat{\mathbb{S}}(x,d+1) \cup \mathbb{S}(x,d+1)$. The entire tree is denoted $\widehat{\mathbb{S}}(x,0)$. We observe that $\forall d : P_i \in \widehat{\mathbb{S}}(x_i,d)$ and $\forall d : P_i \notin \mathbb{S}(x_i,d)$.

In Kademlia, the distance $d(x_i,x_j)$ between two KIDs is defined as their bit-wise XOR interpreted as an integer. In general, a peer $P_i$ with KID $x_i$ stores information on peers with $x_j$ that are close to $x_i$, i.e. for small $d(x_i,x_j)$. For this purpose, $P_i$ disposes of a set denoted $k$-bucket of at most $k$ players $P_j \in \mathbb{S}(x_i,d)$ for every $\mathbb{S}(x_i,d)$ with $d > 0$.[2] See Fig. 1 for an example. The size of subtrees decreases exponentially for growing depth $d$. Consequently, the density of known peers of corresponding $k$-buckets grows exponentially.

Kademlia ensures that the routing table, that is the set of all $k$-buckets, is populated by peer lookup requests for random KIDs to the closest already known peers. Requests are responded with a set of closest, known peers from the routing table. One lookup might require multiple, consecutive request-response cycles.

We assume that peers are either present or absent. Present peers join the Kademlia overlay network within a given time interval and stay responsive until their aggregation step is terminated. The aggregation is carried out in $B$ *epochs*, one tree level at a time. Epochs are loosely synchronized, because peers may have to wait for intermediate aggregates to be computed in order to continue.

First, every peer $P_i$ computes a container for its initial aggregate. The container is assigned to represent the subtree $\widehat{\mathbb{S}}(x_i,B)$ with only \$P_i.

In each epoch for $d = B,\ldots,1$, every peer $P_i$ requests from any $P_j \in \mathbb{S}(x_i,d)$ the aggregate container of subtree $\mathbb{S}(x_i,d)$. $P_j$ responds with the demanded

[2] Note that originally [1] the common prefix length $b$ is used to index $k$-buckets/sibling subtrees while we use the depth $d = b + 1$ of the root of the subtree.

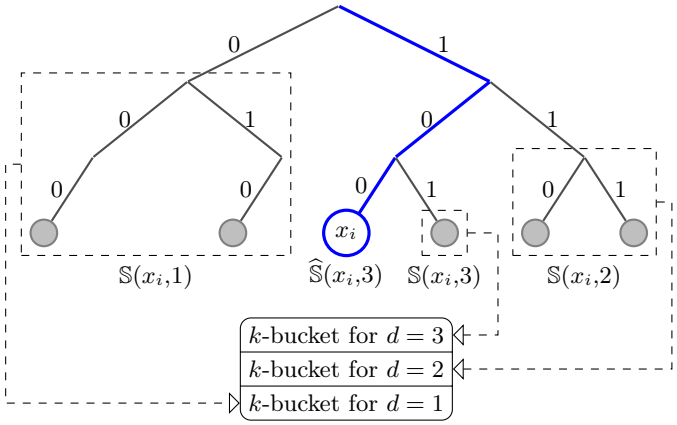



**Fig. 1.** Example of Kademlia $k$-buckets for KID $x_i = 100$ assuming $B = 3$. The sparse tree is partitioned into subtrees $\mathbb{S}(x_i, d)$ with root node at depth $d = 1, 2, 3$. The $k$-buckets for each $d$ contain at most $k$ peers $P_j \in \mathbb{S}(x_i, d)$.

aggregate container. With the received container of $\mathbb{S}(x_i, d)$ and the previously obtained of $\widehat{\mathbb{S}}(x_i, d)$, peer $P_i$ computes the parent aggregate using the aggregation operator $\oplus$. Its corresponding container is then assigned to the parent subtree $\widehat{\mathbb{S}}(x_i, d-1)$. If $\mathbb{S}(x, d) = \emptyset$ for any $d$, the container of $\widehat{\mathbb{S}}(x, d-1)$ is computed only with the aggregate container of $\widehat{\mathbb{S}}(x, d)$ from the previous epoch.

After $B$ consecutive epochs, peer $P_i$ has computed the root aggregate of the entire tree $\widehat{\mathbb{S}}(x_i, 0)$ that contains the initial aggregates of all present peers. If all present peers are honest, the root aggregate is complete and correct.

## 5 Robust Aggregation

The recursive aggregation introduced in Section 4 is very vulnerable to aggregate corruptions leading to erroneous root aggregates, and to illegitimate requests compromising the confidentiality. Following the attack model from [18], we assume a minority of dishonest, Byzantine peers entirely controlled by one adversary that aims to interrupt the aggregation, manipulate root aggregates and increase its knowledge on initial and intermediate aggregates. Byzantine peers can essentially behave arbitrarily, but are assumed to be unable to prevent their initial integration in the routing tables by honest peers.

To prevent Sybil attacks and arbitrary requests, all messages $m$ between peers are signed by the sender $P_i$ using $\sigma_i(m)$ [21]. For signature verification, the public key $pk_i$ and the token $t_i$ must be either published (in the DHT) or sent along with every signature. Henceforth, a peer $P_i$ answers aggregate requests for $\widehat{\mathbb{S}}(x_i, d)$ only for peers $P_j \in \widehat{\mathbb{S}}(x_i, d)$ in the same subtree or $P_j \in \mathbb{S}(x_i, d)$ in the sibling subtree. Consequently, peers cannot obtain more knowledge on aggregates than strictly necessary to compute the root aggregate.

Further, player signatures are employed to detect deviations from the protocol. For every computed aggregate container of $\widehat{\mathbb{S}}(x_i, d)$ with hash $h$ and counter $c$, player $P_i$ produces an aggregate container signature $\sigma_i(h, d, c)$. A signature $\sigma_i(h, d, c)$ expresses the capacity of a peer $P_i$ to compute the container identified by its hash $h$ and is consequently only valid for containers of $\widehat{\mathbb{S}}(x_i, d)$ for any $d$.

In Fig. 2, we consider the steps of $P_j \in \mathbb{S}(x_i, d)$ to produce for any $P_i$ a *confirmed aggregate container* of $\mathbb{S}(x_i, d)$ backed by the signatures listed below. Note that the necessary signatures depend on the subtree configuration that can be explored by $P_i$ using peer lookup requests. Like for the recursive aggregation, $P_j$ requests first the sibling aggregate container (①) if $\mathbb{S}(x_j, d+1) \neq \emptyset$. For $|\mathbb{S}(x_j, d+1)| < k$, the corresponding $k$-bucket is exhaustive [1] and the aggregate counter $c$ must not exceed its size. $k$-buckets are hardened against insertion of false contacts by requiring for all $P_q$ in lookup responses the proof of their KID $(pk_q, t_q)$. Then, the so-called *container candidate* for $\mathbb{S}(x_i, d)$ is computed (②).

New is the *confirmation* (③ and ④) to acquire necessary signatures by otherwise redundant requests to peers in the same subtree $\mathbb{S}(x_i, d)$. Candidates are exchanged solely among peers of that subtree to allow for mutual confirmation.

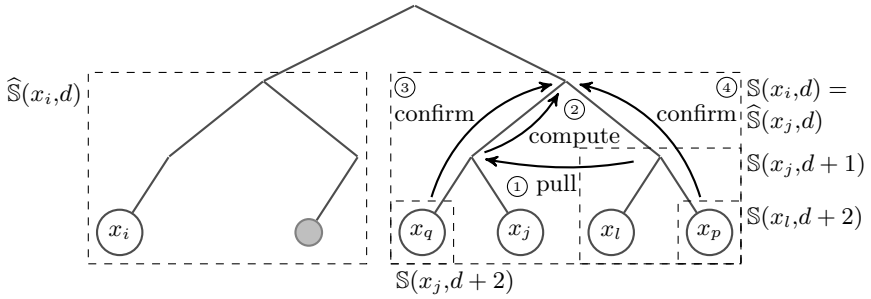



**Fig. 2.** $P_j$ with $x_j$ produces a confirmed aggregate container of $\mathbb{S}(x_i, b)$. This scheme applies to all tree levels with possibly large subtrees to request from.

$P_i$ requires from $P_j$ the following signatures with the container for $\mathbb{S}(x_i, d)$:

1. $P_i$ requires the signature $\sigma_j(h, d, c)$ on container hash and counter.
2. If $c > 1$, there is at least one child aggregate with hash $h_1$ and counter $c_1$ and $\sigma_j(h_1, d+1, c_1)$ must be provided.
3. If $c > 1$ and $c_1 > 1$, a confirmation request (③) is necessary to provide $\sigma_q(h, d, c)$ from $P_q \in \mathbb{S}(x_j, d')$ with the smallest $d' > d+1$ for a non-empty subtree, ideally in the subtree $\mathbb{S}(x_j, d+2)$.
4. If $c > 1$ and $c_2 > 0$, $P_j$ provides $\sigma_l(c_2, d+1, h_2)$ acquired before (①) as 1. signature.
5. If $c > 1$ and $c_2 > 0$, a confirmation request (④) is necessary to provide $\sigma_l(h, d, c)$ if $P_l$ for $c_2 = 1$, and otherwise $\sigma_p(h, d, c)$ from $P_p \in \mathbb{S}(x_l, d')$ with the smallest $d' > d+1$ for a non-empty subtree, ideally in $\mathbb{S}(x_l, d+2)$.

The 1., 2. and 4. signature listed above are required already for candidate containers and allow to detect dishonest peers during confirmation. The 3. and 5. signature promote a consensus in $\widehat{\mathbb{S}}(x_j, d+1)$ respectively $\mathbb{S}(x_j, d+1)$.

The requests ③ and ④ provide additional signatures, that may reveal dishonest peers deviating from the protocol. Note that dishonest peers cannot influence which peers are requested to avoid detection with certainty. For this, we focus on signatures $\sigma_e(h, d, c)$ and $\sigma_e(h', d, c)$ of the same peer $P_e$ with equal counter $c$ for distinct containers ($h \neq h'$) of the same subtree. In case of $c = 1$, $P_e$ derived from the protocol with certainty, is as such detected as dishonest, and its signatures and containers are discarded. A new candidate container without it is computed. The same holds for $c = 2$, because $P_e$ has not discarded itself two distinct containers with $c = 1$ of the same peer, and alike for $c = 3$. Without obvious proof for $c > 3$, we assume $P_e$ to be honest. The discarded signatures form a verifiable proof that is attached to request responses for the newly computed (candidate) container and stored in the DHT under the key $\eta(x_e)$ if there was none before. Detected dishonest peers are permanently removed from the routing table.

With all required signatures, a candidate container of $\mathbb{S}(x_i, d)$ is confirmed and may be requested by peers in $\widehat{\mathbb{S}}(x_i, d)$. If the candidate cannot be confirmed by a peer $P_e$, a proof of former deviation is looked up, and requests to other peers continue for a limited number of tries. If $P_j$ gathers this way a majority of signatures for a different child container than those it has computed earlier, $P_j$ repeats the previous aggregation in order to correct or confirm again its child container. If $P_j$ gathers instead a majority of signatures for a different child container than those it has requested, $P_j$ repeats the current aggregation in order to request potentially a different sibling child container to use. Requests for containers with $c = 1$ are not repeated to prevent revisions of initial aggregates.

The administration step ensures that the global minority of dishonest voters is randomly distributed over the tree. Hence, the implicit majority vote on hashes is supposed to be decided by the local majority of honest peers in the subtree. Note that a vote, and thus a honest majority, is not required for subtrees with less than 4 peers, because dishonest peers are detected and removed based on signatures on containers.

If $P_j$ can still not acquire all signatures, e.g. due to a dishonest peer $P_e$ blocking the confirmation, $P_j$ continues the aggregation nevertheless and compensates the missing signature by both child aggregate containers with all their signatures, so that the aggregate computation of $P_j$ can be reproduced. The confidentiality of $P_j$ and $P_e$ is diminished to the same degree.

At last, the root aggregate container is confirmed by some additional signatures to increase the confidence that it has been adopted by the majority.

## 6 Protocol Properties

Common security properties of online voting protocols [16, 18] are considered using the attack model of a dishonest minority from Section 5.

*Eligibility* The administrator is trusted to sign one authorization request for every eligible peer. Without signature, peers cannot engage in the aggregation.

*Confidentiality* The protocol does not ensure secrecy of the initial aggregate due to the necessity to share it at least once over a pseudonymous channel. However, the access to the initial aggregate is limited to randomly determined peers that acquire mostly partial knowledge, so that confidentiality is ensured to a high degree. The pseudonymous channel between peers augments further the confidentiality. The DHT is ephemeral, distributes information evenly among peers, and vanishes when peers disconnect after the aggregation. Potential data breaches are therefore local and bounded in time.

*Completeness and Correctness* A local majority of dishonest peers in a subtree $\widehat{\mathbb{S}}(x, d)$ with at least 3 peers allow for manipulations of the corresponding aggregate container. Manipulations of its counter $c$ require further at least $k$ peers in $\widehat{\mathbb{S}}(x, d)$. Hence, for a reasonably-sized global dishonest minority, the protocol ensures that peers compute with high probability root aggregates that are with high probability correct or almost correct.

*Verifiability* Using requests, $P_i$ can determine with high probability which root aggregate has been confirmed by most peers and verify the chain of container hashes to the hash of its initial aggregate.

*Robustness and Non-Interruptibility* The aggregation step is entirely distributed to equipotent peers. With no weakest link, the influence of a reasonably-sized dishonest minority is locally limited. The redundancy of the aggregate computation increases exponentially in every epoch as aggregates become more meaningful.

The *protocol complexity* is mostly inherited by the properties of Kademlia, which have been studied [22] and experimentally confirmed as part of BitTorrent.

*Message Complexity* For a network of $n$ peers, a lookup requires with great probability $\mathcal{O}(\log n)$ request-response cycles. Joining the network requires a limited number of lookups and is thus as well of order $\mathcal{O}(\log n)$. With the consideration to estimate the number of empty $k$-buckets from [22], the average number of container requests for the basic aggregation is found to be $\mathcal{O}(\log n)$.

*Memory Complexity* The memory required to store non-empty $k$-buckets is $\mathcal{O}(\log n)$. Further, the aggregation algorithm requires to store $\mathcal{O}(\log n)$ received aggregate containers for non-empty sibling subtrees and perhaps a limited number of alternatives in case of failing confirmations. Hence, for a constant size of aggregate containers, the total memory complexity is again $\mathcal{O}(\log n)$.

*Time Complexity* Intermediate aggregates for ancestor nodes are computed in sequence. For a constant computation time per aggregate and with an upper limit to request and confirm aggregates, the time complexity is $\mathcal{O}(\log n)$.

## 7 Experimental Confidentiality Analysis

The protocol has been simulated on the basis of `kad`, an implementation of Kademlia[3] written in JavaScript with its extension `kad-spartacus`. For each peer $P_i$, key pairs $(pk_i, sk_i)$ are generated using elliptic-curves cryptography. The KID $x_i$ of each peer $P_i$ is derived by hashing $pk_i$ first with SHA-256 and the result again with RIPEMD-160. It is assumed that the use of $pk_i$ instead of the token $t_i$ leads to an equally random distribution of KIDs, so that the administration step can be omitted in the simulation. A simulation parameter allows to vary the generation of key pairs and consequently the KIDs, so that different tree configurations can be tested.

After all $n$ peers are instantiated, every $P_i$ connects to the Kademlia network using an initial contact $P_{i-1}$. According to the Kademlia protocol, peers update their routing table using lookup requests. In our model, peers do not join or leave during the aggregation, so that the routing table does not change hereafter. Once all routing tables are complete, peers start the aggregation step like detailed in Section 4. The simulation does not consider absent or dishonest peers.

If a peer receives a request for an intermediate aggregate that has not yet been computed, the response is delayed. The aggregation steps in the simulation use neither parallel requests nor timeouts for requests.

We consider the issue of confidentiality, and measure both the degree of leakage of initial aggregates, and the concentration of knowledge on initial aggregates. For that purpose, we assume that all initial aggregates are distinct.

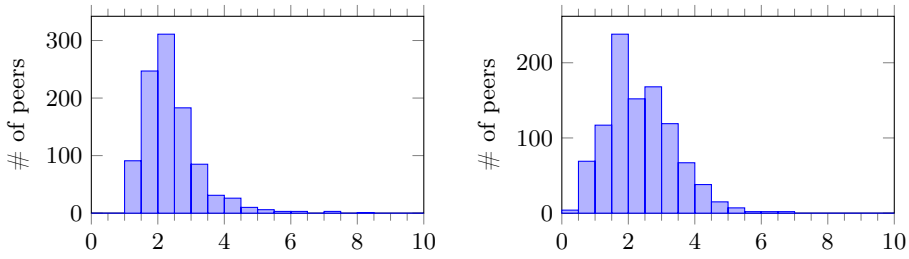
We define the *leaked information* $L_i$ of a peer $P_i$ to be the sum of the inverse of the counters of all containers that $P_i$ used to respond to aggregation requests. $1/c$ denotes the probability to correctly link the contained initial aggregate of $P_i$ to the pseudonym of $P_i$, e.g. an IP address. The leaked information $L_i$ is at least 1, because in a non-trivial aggregation with $n > 1$, $P_i$ must respond at least once with its initial aggregate container with $c = 1$. In a perfectly balanced tree with $n = 2^B$ peers, $L_i$ is strictly smaller than 2:

$$L_i = \sum_{n=0}^{B-1} \left(\frac{1}{2}\right)^n < 2$$

Conversely, we define the *received information* $R_i$ of $P_i$ as the sum of $1/c$ of all containers that $P_i$ receives as responses to its requests. In a perfectly balanced tree, $R_i = L_i$. We further introduce relative measures $l_i = L_i/(n-1)$ and $r_i = R_i/(n-1)$ normed by the worst case that initial aggregates of all other $n-1$ peers are leaked/received. Fig. 3 shows the distribution of $L_i$ and $R_i$ for a simulation run with $n = 1000$ peers. The simulation has been repeated with different tree configurations without notable changes. In the examined case, the relative leak to the network is $l_i = 0.24(9)\,\%$. The relative received information $r_i = 0.24(10)\,\%$ is the same with a slightly higher standard derivation.

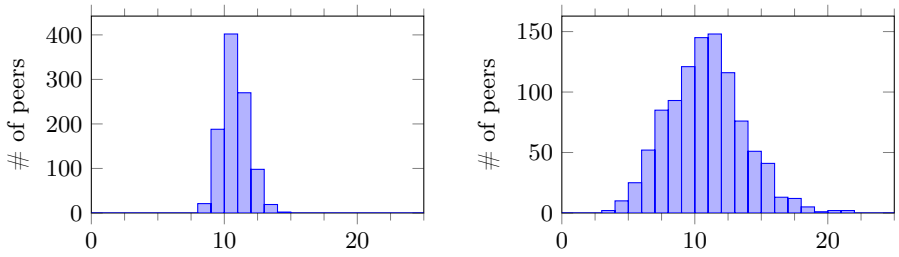
The worst case is given by the least balanced tree configuration in which $|\mathbb{S}(x_i, d)| = 1$ for all $d \in \{B, \ldots, 1\}$. That means for the given $P_i$, every sibling

[3] http://kadtools.github.io/, v1.6.2 released on November 29, 2016




**(a)** Histogram of leaked information $L_i$. **(b)** Histogram of received information $R_i$.

**Fig. 3.** In a simulation with $n = 1000$, peers leak (a), respectively receive (b), information on initial aggregates depending on the global distribution of peers on the binary Kademlia tree. $L_i$ peaks close to the theoretical value 2 of an optimally balanced tree. Only few peers leak significantly more. While the mean for $R_i$ is the same, the distribution is slightly different.




**(a)** Histogram of # of received responses. **(b)** Histogram of # of given responses.

**Fig. 4.** In a simulation with $n = 1000$, the number of given (b) and received (a) responses has been recorded for every peer. While the distribution of received responses is very sharp, the distribution for given responses is twice as broad. In the Kademlia routing tables, some peers are more often represented than others.

subtree contains exactly one other peer. Here, $P_i$ learns in every epoch one initial aggregate with certainty. However, such a tree allows for only $B + 1$ peers and every additional peer decreases $L_i$.

Moreover, the load on peers measured by the number of received and given responses has been examined. The histograms in Fig. 4 indicates that no peer receives significantly more load than others—a property that has been shown for Kademlia before.

Eventually, the average number of requests per peer simulated with different numbers of peers $n$ up to $n = 1000$ confirmed the theoretical message complexity of $\mathcal{O}(\log n)$ shown in Section 6.

## 8 Conclusion

We considered the fundamental problem of large-scale confidential aggregation, and proposed the distributed aggregation protocol ADVOKAT. It prioritizes system wide properties like scalability and robustness over perfect completeness, correctness or full secrecy of initial aggregates.

The aggregation step is distributed to entirely equipotent peers which improves the robustness in face of all sorts of attacks and reduces the reliance on institutional trust. Peers may choose their trusted protocol implementation. Cryptography is only employed to manage authorization and ensure integrity, but not to ensure secrecy, which renders the protocol easier to understand and independent of hardness-assumptions common in cryptography. Due to the even distribution of data and the ephemeral nature of the network, the risk of global or targeted leaks after the aggregation is eliminated. With its global message complexity of $\mathcal{O}(n \log n)$, it outperforms SPP with $\mathcal{O}(n \log n^3)$ [18] and DPol with $\mathcal{O}(n\sqrt{n})$ [19] which both provide instead stronger confidentiality.

We showed that the protocol offers a high level of confidentiality though at least comparable to postal voting with trusted authorities. For large $n$, it is very unlikely that the initial aggregate of a given peer is revealed, which might be acceptable for many applications. Completeness and correctness can be compared to paper-based voting. It is possible that few initial aggregates are manipulated or not counted, but not at a global scale and not often. An individual verification allows to detect manipulations.

The universal protocol algebra supports a wide range of applications, e.g. distributed lottery, aggregation of sensible healthcare data, or all sorts of reduce operations. Turning our protocol into a solution that can be adopted in practice will require some effort. Foremost, a formal definition of completeness and correctness must be introduced so that upper limits of their manipulations depending on the ratio of dishonest peers in the attack model from Section 5 can be formulated. Further, the influence of churn of Byzantine peers on the routing tables must be analysed and, if necessary, restricted to allow for Byzantine peers with no assumptions.

**Acknowledgments** The authors thank Stéphane Frénot, Damien Reimert, Aurélien Faravelon, Pascal Lafourcade and Matthieu Giraud for fruitful discussions on distributed voting protocols and attack vectors.

## Bibliography

1. Maymounkov, P., and Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric. First Intern. Workshop on Peer-to-Peer Systems (2002)
2. Cohen, B.: *The BitTorrent Protocol Specification*, (2008). http://bittorrent.org/beps/bep_0003.html
3. Nakamoto, S.: *Bitcoin: A Peer-to-Peer Electronic Cash System*, (2008). https://bitcoin.org/bitcoin.pdf

4. Reimert, D., Frénot, S., Grumbach, S., and Meyffret, S.: 'Machine de Vote électronique et Infrastructure comportant une telle Machine'. Patent FR3037702 (FR). 2016. http://bases-brevets.inpi.fr/en/document-en/FR3037702.html.
5. Zhang, Z., Shi, S.-M., and Zhu, J.: SOMO: Self-organized metadata overlay for resource management in P2P DHT. In: Peer-to-Peer Systems II, pp. 170–182. (2003)
6. Van Renesse, R., and Bozdog, A.: Willow: DHT, Aggregation, and Publish/Subscribe in one Protocol. In: Peer-to-Peer Systems III, pp. 173–183. (2004)
7. Cappos, J., and Hartman, J.H.: San Fermín: aggregating large data sets using a binomial swap forest. Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (2008)
8. Artigas, M.S., García, P., and Skarmeta, A.F.G.: DECA: A Hierarchical Framework for DECentralized Aggregation in DHTs. Large Scale Management of Distributed Systems (2006)
9. Albrecht, K., Arnold, R., Gähwiler, M., and Wattenhofer, R.: Aggregating information in peer-to-peer systems for improved join and leave. In: Proceedings - 4th Intern. Conf. on Peer-to-Peer Computing, P2P2004, pp. 227–234. IEEE (2004)
10. Evseenko, N.: New hybrid distributed voting algorithm. CoRR abs/1305.0 (2013)
11. Chaum, D.: Blind Signatures for Untraceable Payments. In: Advances in Cryptology: Proceedings of Crypto 82. Ed. by D. Chaum, R.L. Rivest and A.T. Sherman, pp. 199–203. Springer US, Boston, MA(1983)
12. Chaum, D.L.: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communications of the ACM 24(2), 84–90 (1981)
13. Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T.: Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In: Advances in Cryptology — EUROCRYPT '99. Ed. by J. Stern, pp. 295–310. Springer Berlin Heidelberg(1999)
14. Benaloh, J.D.C., and Yung, M.: Distributing the Power of a Government to Enhance the Privacy of Voters. In: Proc. of PODC '86, pp. 52–62. ACM, USA (1986)
15. Fujioka, A., Okamoto, T., and Ohta, K.: A Practical Secret Voting Scheme for Large Scale Elections. In: Seberry, J., and Zheng, Y. (eds.) Advances in Cryptology – AUSCRYPT'92. LNCS, vol. 718, pp. 244–251. Springer, Heidelberg (1993)
16. Ibrahim, S., Kamat, M., Salleh, M., and Aziz, S.: Secure E-voting with blind signature. In: Proc. of NCTT '03, pp. 193–197 (2003)
17. Zhao, Z., and Chan, T.-H.H.: How to Vote Privately Using Bitcoin. IACR Cryptology ePrint Archive (2015)
18. Gambs, S., Guerraoui, R., Harkous, H., Huc, F., and Kermarrec, A.-M.: Scalable and Secure Aggregation in Distributed Networks. arXiv e-prints (2011)
19. Guerraoui, R., Huguenin, K., Kermarrec, A.M., Monod, M., and Vigfsson, Ý.: Decentralized polling with respectable participants. Journal of Parallel and Distributed Computing 72(1), 13–26 (2012)
20. Hoang, B.-T., and Imine, A.: Efficient and Decentralized Polling Protocol for General Social Networks. In: Pelc, A., and Schwarzmann, A.A. (eds.) Stabilization, Safety, and Security of Distributed Systems: 17th International Symposium, pp. 171–186. Springer International Publishing (2015)
21. Baumgart, I., and Mies, S.: S/Kademlia: A practicable approach towards secure key-based routing. In: Proc. of ICPADS '07, pp. 1–8. IEEE Computer Society, Washington, DC, USA (2007)
22. Cai, X.S., and Devroye, L.: A probabilistic analysis of Kademlia networks. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp. 711–721 (2013)