Anonymous contribution of data

Matthew McKague[*] and David Eyers[†]

January 18, 2017

Abstract

Many application domains depend on the collection of aggregate statistics from a large number of participants. In such situations, often the individual data points are not required. Indeed participants may wish to preserve the privacy of their specific data despite being willing to contribute to the aggregate statistics. We propose a protocol that allows a server to gather aggregate statistics, while providing anonymity to participants. Our protocol is information theoretically secure so that the server gains no information about participants' data other than what is revealed by the aggregate statistics themselves.

1 Introduction

Suppose you buy a wearable heart rate monitor. Its software gives you the option to anonymously contribute heart rate data for scientific research. Security conscious individuals are likely to immediately consider the implications of this statement about anonymous participation. Is it possible to achieve anonymous participation? If so, can anonymity be protected in an information theoretic fashion? Is it possible to protect the anonymity without a huge overhead? In this paper we give an affirmative answer to these three questions.

Data contribution is useful in numerous applications — personal health monitoring and fitness systems, smart cities, and other emerging applications

[*]School of Electrical Engineering and Computer Science, Queensland University of Technology. m.mckague@qut.edu.au

[†]Department of Computer Science, University of Otago. dme@cs.otago.ac.nz

from the Internet of Things domain [AIM10, AAS13, McK15]. However, the need to protect privacy may mean that some applications are not possible without measures to ensure anonymity. Healthcare is one field where it may be extremely valuable to collect information for research or clinical purposes, but their can be serious ethical concerns and strict regulatory requirements to protect patients' privacy. In other fields, intelligible guarantees of privacy in and of themselves may help increase participation.

1.1 Previous work

Anonymity has been studied in the context of secure multi-party function evaluation. In secure multi-party function evaluation, n participants each have some input x_j and they wish to evaluate some function $f(x_1 \ldots x_n)$ so that all participants learn the correct value of f, but nothing beyond this. Secure multi-party function evaluation is known to be possible as long as less than a third of participants are dishonest [BOGW88, CCD88], or less than half when a broadcast channel is available [RBO89]. The *dining cryptographers* problem [Cha88] is one early example of an anonymous protocol. The dining cryptographers problem assumes honest participants and attempts to decide whether all inputs are 0 or otherwise.

Broadbent et al. [BT07] developed protocols that preserve anonymity for several problems. Importantly, their protocols do not make assumptions about the number of dishonest participants. All of their protocols are based around a single protocol for evaluating the parity of the participants' input bits. The results assume the presence of a broadcast channel in which messages from all participants are simultaneously announced.

Another aspect of anonymity is exemplified by the Tor onion routing system [DMS04]. Here the idea is — rather than to compute a function from participants' inputs — to hide the identities of participants who are communicating. There may be many conversations going on within the set of participants, but it should be practically impossible to determine which participants are conversing with each other. The degree of protection offered at any point in time will, of course, depend on the number of participants that are active within the Tor network at that time, and whether or not they are colluding.

1.2 Contributions

In this paper we present a straightforward protocol that allows n participants to contribute data to a server such that the server learns the mean of the participants' data points, without leaking any information on any individual participant's data. Security is information theoretic and does not rely on any computational assumptions. The protocol has a communication overhead[1] of $O(\log n)$ and makes no assumption on the number of dishonest participants. However, the overhead does increase with the fraction of dishonest participants.

We then develop further anonymous protocols that build on our initial protocol for:

- sharing the variance and higher statistical moments,
- detecting a consensus,
- detecting a single participant that is breaking consensus,
- contributing single data points,
- establishing anonymous identities and
- contributing time series data.

Our protocols could be adapted for use in a secure multi-party function evolution scenario, provided that there is a simultaneous broadcast channel available. A simultaneous broadcast channel is one where all parties can simultaneously send a message to be received by all other parties. However, our main intention is to contribute data to a designated collection point, such as a single server, in which case no broadcast channels are needed.

2 Anonymous summation channels

Before we describe our protocol, we elaborate on what we wish to accomplish, that is to say, the ideal functionality that we would like to achieve. Suppose that we have a group of participants $\mathcal{P}$ who wish to communicate with a server S. An *anonymous summation channel* is a communication mechanism

[1]the overhead is a multiplicative factor compared to the case of the participants sending their data directly to the server

that will allow each $p \in \mathcal{P}$ to contribute some data $m_p \in \mathcal{M}$ — where $(\mathcal{M}, +)$ is a group — to S such that S learns $M \leftarrow \sum_{p \in \mathcal{P}} m_p$ and nothing else. In particular, we must achieve that S learns nothing about the distribution of m_p for any p other than what would be revealed by knowing M.

We elaborate on how to achieve this end by first specifying the ideal functionality as implemented using a trusted third participant T.

1. Each p sends message m_p to T via a private channel.

2. T sums the messages it receives to form M, which it sends to S via a private channel.

3. T erases all its information.

Some salient features of this channel are:

- The server receives M, and nothing else.

- Participants receive nothing.

- The server could still learn values of m_p for dishonest participants through other communication.

- Participants are in complete control over their contribution. As a consequence, there is no concept of contributing incorrect data.

The last point is especially important for discussing security. A participant may choose to contribute some data that is not meaningful, or is somehow incorrect in a particular context. The ideal channel will still faithfully incorporate this incorrect data. The ideal channel cannot do any better, since it has no other route to "verify" the contribution being made by any participant. For this reason, if there is some behaviour that is equivalent to a participant honestly contributing some data, then we will consider that behaviour to be honest. In particular, any participant may render a run of the protocol meaningless by contributing some meaningless data. Since that is still true of the ideal channel, we will not consider it to be dishonest behaviour in this context. However, it may still be dishonest in a larger context.

Another important point about the ideal behaviour is that anonymity is always with respect to the pool of *honest* participants. In particular, it is possible that an attacker knows that the participants are contributing, perhaps by observing their use of communication channels. Anonymity is for participants' data, not for their participation in the protocol. Also, if there are dishonest participants who are collaborating with the server then their contribution can be removed by subtracting it from M, and anonymity is with respect to the smaller pool of honest participants.

As is often the case, we will not necessarily be able to simulate the ideal functionality perfectly. In particular, with some probability, various conditions may hold that break the security of the protocol. Hence we need to introduce the idea of *approximately anonymous summation channels* where security is achieved except with probability δ, where δ is a tunable security parameter.

3 Description of the protocol

In order to accomplish our task of developing an approximate anonymous summation channel we will need some resources. Our resources will be supplied by the following assumptions:

- There are private authenticated communication channels between each $p \in \mathcal{P}$ and S.
- There are private, authenticated communication channels between all $p, q \in \mathcal{P}$.
- Every $p \in \mathcal{P}$ has private access to a source of random numbers.

3.1 2-participant protocol

Before describing the full protocol that is applicable to any number of participants, we first introduce a restricted variant for two parties. This will give a clear illustration of how the general protocol works.

We start with two participants, p and q, with data m_p and m_q. The protocol is as follows:

1. p chooses a number $r_{p,q}$ uniformly at random from $\mathcal{M}$ and sends it to q over their private authenticated channel.
2. p then sends $r_p \leftarrow m_p + r_{p,q}$ to S.

3. q sends $r_q \leftarrow m_q - r_{p,q}$ to S.

4. S forms $M = r_p + r_q$.

Clearly the protocol is correct as $r_p + r_q = m_p + r_{p,q} + m_q - r_{p,q} = m_p + m_q$.

To see how security is accomplished, first note that if S has M and r_p then this is exactly the same information as if S has r_p and r_q. Indeed, since $r_q = M - r_p$, S can easily go between these two scenarios. So let us look at the second scenario. S has M, as it should, and also has $r_p = m_p + r_{p,q}$. Now $r_{p,q}$ was chosen uniformly at random. Hence we can view r_p as a ciphertext where m_p was encrypted using the one-time-pad with key $r_{p,q}$. This means that r_p contains absolutely no information about m_p or m_q. Thus the only information that S has learned is M.

Each of the two participants has also learned nothing. Indeed, q has only received a random number and nothing else. The only way for anyone to learn information is to collaborate. If q and S collaborate then they can learn p's input. In this case the pool of honest participants is just p, and anonymity is with respect to that pool of size one.

3.2 Full protocol

The full protocol is detailed in Algorithms 1 and 2. A summary of the symbols used appears in Table 3.2. Roughly speaking, we extend the 2-party protocol by having many random numbers sent between many participants. Then when p sends some random number $r_{p,q}$ to another participant q, p adds $r_{p,q}$ to r_p. Whenever p receives a random number, p subtracts it from r_p.

Correctness comes from the fact that when participant p sends $r_{p,q}$ to q, p adds it to r_p and q subtracts it from r_q. Then when S forms M, $r_{p,q}$ will appear twice in the summation, once added and once subtracted. Hence all the $r_{p,q}$ drop out of the summation and M is just the sum of all r_p as required.

4 Proof of security

The proof of security has three main steps. The first step is to prove security in a special case. Secondly, we reduce the case of honest participants down

Symbol	Meaning
$\mathcal{M}$	Message space, that is an additive group
$\mathcal{P}$	Pool of participants
p, q	Participants p and q
S	Server
m_p	data contributed by p
$r_{p,q}$	Random number sent from p to q
r_p	Message sent from p to the server
M	$\sum_{p \in P} m_p$
d	Number of dishonest participants
c	Fraction of dishonest participants (i.e. $d = cn$)
G	Graph of communications between participants
G'	Subgraph of G of honest participants
Q_p	Participants that p sends random numbers to
R_p	Participants that p receives random numbers from
k	Number of participants to send messages each participant exchanges random numbers with

Table 1: Symbols used in the algorithm listings

Algorithm 1 Honest participant protocol for participant p.

```
1: procedure SETUP($m_p$)
2:     $r_p \leftarrow m_p$
3: end procedure
4:
5: procedure SEND MESSAGES
6:     Choose a set $Q_p$ of size $k$ randomly from $\mathcal{P} \setminus \{p\}$
7:     for $q \in Q_p$ do
8:         Randomly choose $r_{p,q}$ from $\mathcal{M}$
9:         Send $r_{p,q}$ to $q$
10:        $r_p \leftarrow r_p + r_{p,q}$
11:    end for                          ▷ The value $r_{p,q}$ should not be retained
12: end procedure
13:
14: procedure MESSAGE RECEIVED($r_{q,p}$)        ▷ Asynchronous event handler
15:     If $r_{q,p}$ is malformed, then ignore it.
16:     $r_p \leftarrow r_p - r_{q,p}$
17: end procedure                       ▷ The value $r_{p,q}$ should not be retained
18:
19: procedure FINAL
20:     Send $r_p$ to $S$
21: end procedure
```

Algorithm 2 Honest server protocol

```
procedure SETUP
    $M \leftarrow 0$                          ▷ Should finish as $M$ defined in Table 3.2
    $c_m \leftarrow 0$                        ▷ Count of messages received
    $P \leftarrow \emptyset$                  ▷ Accumulates which $p$ have sent $r_p$
end procedure

procedure MESSAGE RECEIVED($r_p$)
    $M \leftarrow M + r_p$
    $c_m \leftarrow c_m + 1$
    $P \leftarrow P \cup \{p\}$
end procedure

procedure FINAL
    if $c_m \neq \|\mathcal{P}\| \vee P \neq \mathcal{P}$ then
        abort
    else
        output $M$
    end if
end procedure
```

to the special case. Finally, we consider dishonest participants and reduce down to the honest participant case.

Before we get into the proofs, let us set down some basic facts and definitions. First, for a participant p, define R_p to be the set of participants that have sent a random number to p. That is, $R_p = \{q \,|\, p \in Q_q\}$. Then the message that p sends to the server is

$$r_p = m_p + \sum_{q \in Q_p} r_{p,q} - \sum_{q \in R_p} r_{q,p} \tag{1}$$

Next, we form a directed graph G such that (p, q) is an edge if $q \in Q_p$. We will assume that after the protocol is run G becomes public knowledge, so that privacy is not dependent on secrecy around who is communicating with whom. Note that G is generated randomly, and independently from the data, so it does not contain any information about the participants' data.

4.1 Honest case

For now we will assume that all participants are honest and that S is honest but curious. That is to say, we are only concerned with what information S receives from the protocol. We want to show that S does not learn anything beyond M. In the first case, we consider a variant of the protocol where the graph G is fixed beforehand rather than chosen randomly by the participants.

Lemma 1. *If G is a directed rooted tree and all participants are honest, then the server learns exactly M.*

Proof. First, some notation. Set R to be the root of the tree, P_p to be the parent of p, C_p the set of children of p and D_p the set of all descendants of p. Then the messages received by the server from each participant p are

$$r_p = m_p - r_{P_p,p} + \sum_{q \in C_p} r_{p,q}. \tag{2}$$

The root R has no parent, so we leave $r_{P_R,R}$ off for that special case.

Now we will reduce to a different protocol where S receives different messages. This idea is a generalisation of how we proved security in the 2-participant case. Suppose that the server receives

$$r'_p = r_p + \sum_{q \in C_p} r_q \tag{3}$$

for each p. Note that these values are easily derived from the messages that S receives in the original protocol. Also, given r'_p for all p, S can calculate each r_p. To see this first note that $r'_p = r_p$ for any leaf vertex p since $C_p = \emptyset$ in that case. Now supposing that r_q is known for all children q of p, we can subtract them off r'_p to get r_p. Thus we can apply induction starting at the leaf vertices and obtain r_p all the way up the tree. This means that the set of all r'_p contains exactly the same information as the set of all r_p.

Let us prove another property. For the leaf vertices p we have

$$r'_p = m_p - r_{P_p,p} + \sum_{q \in D_p} m_q \tag{4}$$

since D_p is empty. Now suppose that the above formula is true for all the descendants of some vertex p. Then applying the formula along with the definition of r'_p we find

$$\begin{aligned} r'_p &= r_p + \sum_{q \in C_p} r_q && (5)\\ &= m_p - r_{P_p,p} + \sum_{q \in C_p} r_{p,q} + \sum_{q \in C_p} r_q && (6)\\ &= m_p - r_{P_p,p} + \sum_{q \in C_p} \left[r_{p,q} + m_q - r_{p,q} + \sum_{s \in D_q} m_s \right] && (7)\\ &= m_p - r_{P_p,p} + \sum_{q \in D_p} m_q && (8) \end{aligned}$$

Applying induction all the way up we find the same is true for all p other than R. For the root R we leave off the $r_{P_p,p}$ since R has no parent. Hence $r'_R = M$. This establishes that S can obtain M from all of the r_p's.

Now we see that each r'_p is the sum of some data with an independently chosen random number $r_{P_p,q}$. Thus each r'_p is effectively one-time-padded with its own independent key. The exception is r'_R, which is just M. Thus what the server has is just M and a collection of ciphertexts that are completely independent of each other. We have thus shown that the protocol leaks no information other than M. $\square$

Now we move from the special case to the original protocol where G is determined randomly. We do this by showing that when G is weakly connected we can reduce security to the case where G is a rooted tree. Essentially we find a spanning tree and argue that the operations required to turn G into a root tree do not leak any information to S.

Lemma 2. *Assume all participants are honest. If G is a weakly connected graph, then the server learns exactly M.*

Proof. G is weakly connected, so there exists a spanning tree T of the underlying undirected graph of G. Suppose we announce all $r_{p,q}$ where (p, q) is in G but $\{p, q\}$ is not an edge in T. This only increases the information that the server has. In fact, whenever we announce $r_{p,q}$ the server can add this to r_p and subtract it from r_q after which the server's information is exactly the same as if $r_{p,q}$ was never sent between p and q. So announcing $r_{p,q}$ is the same as deleting the edge (p, q) from G.

Next, to get a rooted directed tree G', we choose some vertex R in T to be the root and orient all edges from parent to child. Some of these edges may be backwards compared to the original edge in G. However, we can flip an edge around by noting that an edge going backwards is the same as replacing $r_{p,q}$ with $-r_{p,q}$. From the server's perspective these are both uniformly random, so there is no change in the server's information by making this change. If both (p, q) and (q, p) are in G, then we can announce whichever one is not in G'.

To summarise, for each $(p, q) \in E(G)$ we do the following:

- If neither (p, q) nor (q, p) are in $E(G')$ then announce $r_{p,q}$.
- If $(q, p) \in G'$ and $(q, p) \in G$ then announce $r_{p,q}$.
- If $(q, p) \in G'$ but $(q, p) \notin G$ then set $r_{q,p} = -r_{p,q}$.
- If $(p, q) \in G'$ then do nothing.

G' is now a rooted directed tree, and so the previous lemma applies. The reduction has only announced more information to the server, so the information available when the graph is G is again exactly M. The announced information is just a set of random numbers that are uncorrelated with the participants' data and the other random numbers. $\square$

4.2 Dishonest participants

For greatest generality, we assume that all dishonest participants are collaborating with each other and with the server. This will provide an upper bound on the capabilities of any set of adversaries.

We might also consider whether dishonest participants could launch an attack against an honest server, disrupting service, leading the server to accept an incorrect outcome, or learning information that only the server would normally obtain. The first two attacks are clearly possible by disrupting communication channels, or by contributing inappropriate data according to the honest algorithm. Both of these problems are outside the scope of our protocol. As for information leakage, the only information (other than ciphertexts for private communications) that dishonest parties receive are random numbers which are not correlated with any data. Hence dishonest parties will not know any information about the honest participants' data.

For the remainder of this section we assume that there is a set D of participants who are dishonest and collaborating with a dishonest server. We suppose that they are all operating according to a common strategy and sharing all information. This provides an upper bound on all other strategies.

What can dishonest participants do? They could:

- send more than one message to a participant;
- send messages to more or fewer than k other participants, or choose other participants not uniformly at random;
- choose messages not uniformly at random; or
- send malformed messages.

Let us deal with the first item. Every message received by an honest participant p is simply added into the sum for r_p. Now if a dishonest participant q sends honest participant p several messages $r^{\ell}_{q,p}$ for various ℓ, this is equivalent to sending one message, $\sum_{\ell} r^{\ell}_{q,p}$. Hence there is no gain in sending more than one message, and we will assume that only one message (if any) is sent.

If a dishonest participant changes their method of choosing who they will send messages to, then this will affect the structure of G. However, it will not affect the subgraph of honest participants since by definition these edges are not between honest participants. Our security argument will look only at the subgraph of honest participants, so how the dishonest participants choose their neighbours will not affect security.

A dishonest participant might send a valid looking message which has not been chosen uniformly at random as dictated by the protocol. However, as we will soon argue, S's information will always be the same as if dishonest participants send no messages at all. Hence the distribution of messages is not important.

Finally, if dishonest parties send malformed messages then the recipient will ignore them. With all of these considerations, we will thus suppose that each dishonest participant sends one well formed message to each of some set of other participants.

Now, for some honest p, the server knows all the messages between p and the dishonest participants. Hence the server can subtract $r_{p,d}$ and/or add $r_{d,p}$ to r_p for each dishonest d that has communicated with p. This forms a new r_p which is identical to what p would send if it had not communicated with any dishonest participants. There is no additional information in $r_{p,d}$ or $r_{d,p}$ since these were chosen without reference to p's data. Thus the server's information is the same as if the dishonest participants did not communicate with any honest participants. Equivalently, we can remove all dishonest participants from G.

We have considered the possible actions of the dishonest participants and concluded that we can equivalently consider the case where the dishonest participants are removed from the protocol. As we have previously argued, having a graph G of participants which is connected is important. In this context we need the graph to be connected even after removing all dishonest participants. If communication is never disrupted, as in the honest case, then it suffices to have a weakly connected graph. However, when we later argue for security in the presence of communication disruptions we will need G to be *strongly* connected. Hence we would like to know that G is strongly connected except with some small probability, as show in the following lemma.

Lemma 3. *Let a directed graph G on n vertices V be generated by a random process where each vertex chooses k neighbours uniformly at random. Let $D \subset V$ be a subset of vertices with $|D| \leq cn$ for some $0 \leq c < 1$. Then the probability that the subgraph of G on $V \setminus D$ is not strongly connected is bounded above by*

$$4n\left(\frac{1+c}{2}\right)^{k}. \tag{9}$$

Proof. Set D to be the set of dishonest vertices and set G' to be the induced subgraph of G on $V \setminus D$. It is an easy argument to see that G' is strongly connected if and only if for every partition (A, B) of the vertices of G' (with A and B not empty) there exists at least one edge from A to B.

Our argument will be to sum, over all possible partitions (A, B), the probability that there are no edges from A to B or B to A, assuming that $|A| \leq |B|$. This will provide an upper bound on the probability that G' is not strongly disconnected. To this end, let (A, B) be a partition of $V \setminus D$ and set $a = |A|$, $b = |B|$, and $d = |D|$, so $n = a + b + d$.

Looking at partition (A, B), there are three things which might happen to make G' disconnected. There might be no edge from A to B, no edge from B to A, or no edges at all between A and B. These events are not independent, and to upper bound the probability that there is a disconnection between A and B we just need to sum the probability that there is no edge from A to B, and the probability that there is no edge from B to A. This in fact double counts the cases where there are no edges at all.

If there are no edges from A to B then each individual participant in set A has chosen k other vertices from the $a + d - 1$ vertices not in B. The probability of this happening is

$$\left(\frac{\binom{a+d-1}{k}}{\binom{n-1}{k}} \right)^{a}. \tag{10}$$

Expanding this out and using the fact

$$\frac{s - \ell}{t - \ell} \leq \frac{s}{t} \tag{11}$$

for $0 \leq \ell < s \leq t$ we can upper bound the probability by

$$\left(\frac{a + d}{n} \right)^{ak}. \tag{12}$$

If there are no edges from B to A then each participant in B has chosen k neighbours from the $n - a - 1$ other vertices not in A. The probability of this happening is bounded above by

$$\left(\frac{\binom{n-a-1}{k}}{\binom{n-1}{k}} \right)^{b} \leq \left(\frac{n - a}{n} \right)^{bk} \leq \left(\frac{1 - \frac{a}{n}}{n} \right)^{kn\frac{1-c}{2}} \tag{13}$$

where we have used the fact that $b \geq \frac{n-d}{2}$. Now $(1+1/x)^x$ is always less than e for all real x. Using this we can easily prove that $(1-a/n)^n \leq e^{-a}$, so the above is bounded above by

$$e^{-ak\left(\frac{1-c}{2}\right)} \tag{14}$$

Now consider all possible partitions into A and B. Summing the probability of being disconnected over each partition provides an upper bound on the probability of G' not being strongly connected:

$$\sum_{a=1}^{(n-d)/2} \binom{n-d}{a} \left(\left(\frac{a+d}{n} \right)^{ak} + e^{-ak\left(\frac{1-c}{2}\right)} \right) \tag{15}$$

Straightforwardly,

$$\binom{n-d}{a} \left(\frac{a+d}{n} \right)^{ak} \leq n^a \left(\frac{a+d}{n} \right)^{ak} \leq \frac{(a+d)^{ak}}{n^{a(k-1)}} \tag{16}$$

With $d = cn$ and $a \leq (n-d)/2$, we further bound this above by

$$\left(\left(\frac{1+c}{2} \right)^k n \right)^a. \tag{17}$$

Provided that $\left(\frac{1+c}{2}\right)^k n \leq \frac{1}{2}$, summing over a is a geometric series which is upper bounded by

$$2n \left(\frac{1+c}{2} \right)^k. \tag{18}$$

To bound the second term in (15) we use

$$\binom{n-d}{a} e^{-ak\left(\frac{1-c}{2}\right)} \leq \left(ne^{-k\left(\frac{1-c}{2}\right)} \right)^a. \tag{19}$$

Again, when $e^{-k\left(\frac{1-c}{2}\right)} \leq \frac{1}{2}$ we can bound the sum over a using a geometric series, and the above is bounded by

$$2ne^{-k\left(\frac{1-c}{2}\right)}. \tag{20}$$

Now (15) is bounded above by

$$2n \left(e^{-k\left(\frac{1-c}{2}\right)} + \left(\frac{1+c}{2} \right)^k \right). \tag{21}$$

A bit of calculus shows that for all $0 \le c \le 1$ we have $e^{-\frac{1-c}{2}} \le \frac{1+c}{2}$, so we can simplify this to our final bound of

$$4n\left(\frac{1+c}{2}\right)^k. \tag{22}$$

For this bound to be valid we require that k is large enough so that

$$ne^{-k\left(\frac{1-c}{2}\right)},\ n\left(\frac{1+c}{2}\right)^k \le \frac{1}{2} \tag{23}$$

but this is the same as saying that our final bound is less than 1, i.e. the final bound is a non-trivial bound on probability. □

The lemma tells us that, however many dishonest participants there are, we can choose k large enough so that the probability that the subgraph of honest participants is not strongly disconnected is below whatever threshold we desire. Moreover, $k = O(\log n)$ is sufficient.

Now, let G' be the subgraph of G of honest participants. As we have argued, any information that the server gains from the dishonest parties is equivalent to removing all dishonest edges from G, leaving us with G'. By Lemma 2, whenever G' is weakly connected the server learns exactly M and no other information.

There is one remaining possibility, which is that the server (or some agent of the server) disrupts communication between participants somehow. Perhaps the message is corrupted so that authentication fails or the message is lost entirely. Then the recipient will not process the message. Suppose that p sent $r_{p,q}$ to q, but q did not receive it. Further suppose that r_p and r_q are the messages that would have been sent to the server in the case that that $r_{p,q}$ had never been sent. Then with the disrupted message q still sends r_q to S, but p now sends $r_p + r_{p,q}$ since p adds $r_{p,q}$ regardless. This is equivalent to removing the edge (p, q) from G' and p contributing $m_p + r_{p,q}$. Note that $r_{p,q}$ will remain unknown to the dishonest participants.

Suppose now that some number of messages have been lost. With very low probability G' was not strongly connected, and we have nothing to say about security in that case, so let us assume that G' is strongly connected. Now let G'' be the graph where we have taken G' and removed all edges where the message was lost. If G'' is still weakly connected then the security reduction still holds. So let us suppose that G'' is not weakly connected and let H

be some weakly connected component of G''. The same security argument still applies to H, so that the server learns exactly $\sum_{p \in H} m_p$. However, by assumption G' was strongly connected, there was some $v \in H$ that sent a message to some q outside of H. This message was lost, and as we have argued v effectively contributes $m_v + r_{v,q}$, meaning that the server actually learns $\sum_{p \in H} m_p + r_{v,q}$. This again looks like a one-time-padded message, so the server has learned nothing at all about the data from participants in H. The same argument applies to all connected components. Thus any attacks that drop or corrupt messages will only reduce the amount of information that the server learns.

Finally, we combine the above results to be able to state our security theorem.

Theorem 1. *If $d \leq cn$ then, except with probability at most $4n\left(\frac{1+c}{2}\right)^k$, the server learns at most M and no participant learns anything.*

Suppose we set a security constant of δ so that we want to have security except with probability at most δ. Then given n we can choose k that gives us security δ. Taking d to be a pessimistic value of $\frac{1}{2}n$ we find that to achieve security δ it suffices to set

$$k = \lceil 2.41\left(\log_2 n + 2 - \log_2 \delta\right)\rceil. \tag{24}$$

5 Extensions

For many applications, the server will not only be interested in the sum of the participants' data. In this section we document some possible uses of our protocol to provide additional functionality. In all cases, honest participants' anonymity is protected.

5.1 Means, variance, and other moments

Suppose that the server would like to find out about the distribution of the participants' data, but not individual data points. Using our protocol it is very straightforward for the server to learn exactly $\sum m_p$, from which the mean is easy to find. Once the mean is known, it is easy to find the variance if the participants all contribute $(m_p)^2$ so that the server learns $\sum (m_p)^2$.

Indeed, the variance is found via the formula

$$\sigma^2 = \frac{1}{n}\left(\sum_p m_p^2\right) - \frac{1}{n}\left(\sum_p m_p\right)^2 \tag{25}$$

where the two sums inside parentheses can be straightforwardly computed anonymously using our protocol. Note that if the server learns the mean and variance then it can also compute $\sum_p m_p^2$ by rearranging the above formula, so this does not leak any more information than is necessary.

Higher moments μ_t can be straightforwardly computed if the server announces the mean μ and the participants contribute $(m_p - \mu)^t$. The formula is given by

$$\mu_t = \frac{1}{n}\left(\sum_p (m_p - \mu)^t\right). \tag{26}$$

where the server learns the sum inside the parentheses in one run of the protocol, after first learning μ.

These techniques allow the server to learn descriptive statistics without revealing any particular data points.

5.2 Consensus

Suppose that the participants voluntarily assign themselves to either set A or set B. The server would like to find out whether set B is empty or not[2]. We can accomplish this by an anonymous contribution protocol where we set

$$m_p = \begin{cases} 0 & p \in A \\ r \in_R \mathcal{M} & p \in B \end{cases}. \tag{27}$$

Suppose that the server learns $M = \sum_p m_p$. If $M \neq 0$ then the server concludes that there is at least one $p \in B$. If $M = 0$ then the server concludes that $|B| = 0$.

How does this protocol perform? If all participants are honest and $|B| = 0$ then M will always be 0. If $|B| \geq 0$ then M will be uniformly distributed and will be non-zero except with probability $\frac{1}{|\mathcal{M}|}$. Similarly, if there are multiple

[2]This is a variant of the Dining Cryptographers problem where the server learns the outcome rather than all participants

honest participants in B then M will be uniformly random and the fact that $|B| \neq 0$ will be detected with high probability.

This protocol has reasonably good resistance to dishonest participants. If there is an honest $p \in B$ then M will be uniformly random regardless of what the other participants send. So a dishonest participant cannot prevent the server from detecting that $|B| \neq 0$, except with probability $\frac{1}{|\mathcal{M}|}$, the same as the honest case. If the dishonest participant wants the server to believe that $|B| \geq 0$ then they can send a random number, but this is the same as the honest behaviour for assigning themselves to B! Hence we do not count this behaviour as dishonest.

One potential dishonest behaviour for this protocol is that if there is more than one dishonest participant then they can conspire so that the sum of their contributions equals 0, but each contribution is non-zero. From the server's point of view this is equivalent to all of the dishonest participants being in A since the outcome for the server will be the same. From a semantic point of view, the behaviour is not problematic since there is an honest behaviour that produces exactly the same outcome. Namely, the dishonest parties could just assign themselves to A and behave honestly.

5.3 Counting

The server can count the number of participants in B if participants contribute 1 when they are in B. However, dishonest participants can very easily manipulate this version of the protocol by contributing negative or large positive numbers.

5.4 Exactly one

Suppose we want to know if B contains exactly 1 participant, no participants, or otherwise. The counting protocol works, but it is disrupted easily by dishonest participants. We can fix this by having honest participants contribute messages in a particular format. Suppose that f is a random oracle.

Participants contribute[3]

$$\begin{cases} (0,0) & p \in A \\ (r, f(r)),\ r \in_R \mathcal{M} & p \in B \end{cases}. \tag{28}$$

If the server observes that M has the form $(a, f(a))$ then it concludes that $|B| = 1$. If $M = (0,0)$ then the server concludes $|B| = 0$ and otherwise the server rejects. As we shall see, rejecting means either there are multiple participants in B or there exists a dishonest participant. Let us call this protocol *exactly one*.

Let us look at the honest case first. There are six possible cases in which the server's conclusion does not match the inputs. Two of these cannot occur, namely when all inputs are $(0,0)$ and the server concludes either $|B| = 1$ or rejects. If $|B| = 1$ then the first bad case is $M = (0,0)$ which occurs only when $r = 0$ (probability $\frac{1}{|\mathcal{M}|}$). If $f(0) \neq 0$ then this case never occurs. The second bad case is when $M = (a,b)$ with $b \neq f(a)$, which does not occur in the honest case. If $|B| > 1$ the two bad cases are $M = (0,0)$ and $M = (a, f(a))$ for some a. When $|B| > 1$ then in the honest case $M = (a,b)$ with a uniformly random. When f is a random oracle b will also be uniformly random. Hence the first bad case happens with probability at most $\frac{1}{|\mathcal{M}|^2}$. The second bad case happens only when two uniformly random numbers are equal, which happens with probability $\frac{1}{|\mathcal{M}|}$.

Let us now look at the dishonest case. This protocol is vulnerable to the same attack as the consensus protocol, namely dishonest participants conspiring so that their contributions sum to $(0,0)$ or $(a, f(a))$. The first case is equivalent to all the dishonest participants instead assigning themselves to A. The second case is equivalent to a single dishonest participant in B and the rest in A. Both of these cases correspond to honest behaviours so we will not consider them.

Suppose that the sum of the dishonest participants' contributions is (a,b) with one of a, b not equal to 0 and $b \neq f(a)$. In this case the server should always reject. We will take the convention here that A and B contain no dishonest participants. If $|B| = 0$ then $M = (a,b)$ and the server rejects since $b \neq f(a)$. If $|B| = 1$ then $M = (a + r, b + f(r))$. The server will incorrectly conclude $|B| = 0$ if $a = -r$ and $b = -f(r)$, which happens with

[3] The notation (a,b) implies that there are two runs of the protocol, and the server considers the information together. Equivalently, there is one protocol which runs over the group $\mathcal{M} \times \mathcal{M}$.

probability $\frac{1}{|\mathcal{M}|^2}$. In other cases the server correctly decides that $|B| = 1$, or rejects. If $|B| > 1$ then $M = (c, d)$ where c and d are randomly distributed. The server will be wrong only if $(c, d) = (0, 0)$ (probability $\frac{1}{|\mathcal{M}|^2}$), or $d = f(c)$ (probability $\frac{1}{|\mathcal{M}|}$).

For all cases, the worst probability that the server is wrong is $\frac{2}{|\mathcal{M}|}$.

5.5 Single data points

We can extend the *exactly one* protocol to allow the single participant in B to contribute some data. This allows the server to determine that exactly one participant has contributed data, without knowing who that participant is. Each participant p contributes

$$\begin{cases} (0,0,0) & p \notin B \\ (r, m_p, f(r+m_p)),\ r \in_R \mathcal{M} & p \in B \end{cases} \tag{29}$$

The server learns $M = (a, b, c)$. If $a = b = 0$ then nobody is in B. If $c = f(a+b)$ then the server takes b as the data. If $c \neq f(a+b)$ then the server rejects. Let us call this protocol *single data*.

Following a similar logic to the *exactly one* protocol, if there is a dishonest party or more than one participant contributing, then M will with high probability not have the form $(a, b, f(a+b))$ or $(0, 0, 0)$ and the server will reject.

5.6 Anonymous identities and time series data

For some applications the server may wish to know individual data points, and may wish to associate data points together over time. One way of achieving this functionality is to have identities, which are attached to some participant, but in such a way that only the participant knows their identity.

One way to establish such identities is to use the *exactly one* protocol to claim identities. Suppose that the server announces "Who would like to be participant x". Then the server runs the *exactly one* protocol. If there is exactly one participant in B, then that participant becomes participant x and the server moves on to the next identity. If B is empty or the *exactly one* protocol rejects, the the server repeats. Supposing that participants attempt to claim an identity with probability about $\frac{1}{2m}$ where m is the number of participants who have not yet claimed an identity, this protocol will not need too many retries. Indeed with this scheme the probability of needing a retry can be bounded above using Bernoulli's inequality by 0.75.

Another way to claim identities would be to use the *single data* protocol. Here the data that a participant contributes could be the requested identity.

Once identities are established, they can be used to establish time series data. For example, the server may announce "Participant x please contribute data" followed by a run of the *single data* protocol. Or the server may run the *single data* protocol without first specifying an identity. Then a participant who wishes to do so can contribute a string containing their identity and their data.

It is possible to establish a private channel from the server to an anonymous identity. The server can request that participant x send a random string using the *single data* protocol. If it succeeds then the server uses the random string to encrypt a message using the one-time-pad, and then broadcasts the ciphertext. Only participant x knows the random string and can decipher the message.

Note that it is easy to disrupt someone attempting to use an identity by contributing garbage whenever the server requests data from a particular identity. However, it is not possible to steal an identity as long as the original owner of the identity continues to participate in the protocols. The *exactly one* and *single data* protocols will reject whenever two or more participants try to contribute. If the owner of an identity drops off the network or stops responding correctly when the server requests information from their identity, then a dishonest participant may take up that identity. If the owner is known to leave the network then the server may notice that a particular identity stops responding and conclude that the leaving participant owned that identity. Hence maintaining both the integrity and confidentiality of an anonymous identity requires continued participation in the network.

5.7 Peer to peer communication

So far we have designated a server S who learns M after a run of our protocol. However, we can instead use a simultaneous broadcast channel, as in [BT07]. In this version participants broadcast r_p instead of sending it to the server. In this way all participants learn M after a run of the protocol. We can use this idea to build a network for anonymous peer to peer communication. A sketch of one possible protocol is as follows:

1. Everyone claims an identity using the *exactly one* protocol.
2. Using the *single data* protocol, a sender announces a recipient. Repeat until the protocol accepts, at which point the recipient is set.
3. Using the *single data* protocol, the recipient distributes a random key. Repeat until the protocol accepts.
4. Using the *single data* protocol, the sender announces the message added to the random key. Repeat until the protocol accepts.
5. Go back to 2 for the next message

Since the protocol is susceptible to disruption by dishonest parties, it is straightforward for an attacker to launch a denial of service attack. Also, it is important that the broadcast channel be simultaneous. If the channel instead operates so that one participant learns all the other r_p's before it needs to send its own r_p, then it can choose a value for r_p that gives any desired M.

5.8 Improving communication efficiency

The protocol, as written, requires many random numbers to be transferred over authenticated private channels. The private channels could be implemented in many ways, one of which is using a one-time-pad along with a message authentication code. These primitives already imply shared randomness at least as long as the random numbers that need to be transferred. Hence one possible improvement to the protocol is to consume the shared randomness directly rather than sending new random numbers. In this case the messages no longer need to be private, and can be much shorter. The shortest length would be one that allowed the recipient to identify the sender, perhaps with some additional information identifying which key to use.

If information theoretic security is not required, then various primitives with computational security could drastically reduce the overall communication required, and eliminate the need for pre-shared keys. One obvious method would be to use Diffe-Hellman key exchange as a source of random numbers instead of pre-shared keys. Also, participants need not choose new neighbours on every run of the protocol, so a pseudo-random number generator could provide a long stream to be used over many runs of the protocol.

Various certificate schemes could be used to provide authentication. All of these methods can reduce the overhead of communication required to run the protocol.

6 Discussion

Our protocols are in many ways very similar to those in [BT07]. Our basic protocol, the anonymous summation channel, is in some ways an extension of Broadbent et al.'s parity protocol to arbitrary abelian groups. However, we offer the advantage of a much lower overhead — $O(\log n)$ compared to $O(n)$ — since our protocol has participants share random numbers with a subset of other participants. As well, for our intended applications we can replace the simultaneous broadcast channel with a designated server. Broadbent et al. develop a similar list of protocols to ours, and in many cases our protocols are again natural generalisations to larger groups. Importantly, our protocols can be used to gather meaningful statistical information, such as mean, variance and higher moments, which is useful for our intended applications in anonymous data contribution.

In terms of functionality, a comparison with `TOR` [DMS04] and onion routing in general is also apt. However, the protocols involved are very different in this case. Our protocol is designed for collecting aggregate information while preserving privacy for individuals contributing data points. In contrast, onion routing is concerned with preserving anonymity for communication between peers. Onion routing could also be used to simulate an anonymous summation channel. A participant can choose a random number r and send $m_p + r$ and $m_p - r$ to the server. A suitable onion routing protocol would hide the source of these two messages from the server. Provided that many participants are contributing similar messages, the server could reconstruct M, but not an individual m_p. This type of protocol seems to require public key encryption, otherwise the server would need to know who sent a message in order to choose which symmetric key is required for description.

References

[AAS13] Charu C Aggarwal, Naveen Ashish, and Amit Sheth. The Internet of Things: A survey from the data-centric perspective. In

Managing and mining sensor data, pp. 383–428. Springer, 2013.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer networks*, **54**(15):2787–2805, 2010.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 1–10. ACM, 1988.

[BT07] Anne Broadbent and Alain Tapp. Information-theoretic security without an honest majority. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, *Lecture Notes in Computer Science*, volume 4833, pp. 410–426. Springer, 2007.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 11–19. ACM, 1988.

[Cha88] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, **1**(1):65–75, 1988.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004. EPRINT http://www.dtic.mil/dtic/tr/fulltext/u2/a465464.pdf.

[McK15] McKinsey Global Institute. The Internet of Things: mapping the value beyond the hype, 2015.

[RBO89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pp. 73–85. ACM, 1989.