PrivEx: Private Collection of Traffic Statistics for Anonymous Communication Networks*

Tariq Elahi
Cheriton School of Computer Science
University of Waterloo,
Waterloo, ON, Canada
tariq.elahi@uwaterloo.ca

George Danezis
Dept. of Computer Science
University College London
London, United Kingdom
g.danezis@ucl.ac.uk

Ian Goldberg
Cheriton School of Computer Science
University of Waterloo,
Waterloo, ON, Canada
iang@cs.uwaterloo.ca

ABSTRACT

In addition to their common use for private online communication, anonymous communication networks can also be used to circumvent censorship. However, it is difficult to determine the extent to which they are actually used for this purpose without violating the privacy of the networks' users. Knowing this extent can be useful to designers and researchers who would like to improve the performance and privacy properties of the network. To address this issue, we propose a statistical data collection system, PrivEx, for collecting egress traffic statistics from anonymous communication networks in a secure and privacy-preserving manner. Our solution is based on distributed differential privacy and secure multiparty computation; it preserves the security and privacy properties of anonymous communication networks, even in the face of adversaries that can compromise data collection nodes or coerce operators to reveal cryptographic secrets and keys.

1. INTRODUCTION

Anonymity on the Internet provides the means to dissociate one's network identity from one's online activities and communications. Anonymity is not offered by default on today's Internet and requires the use of overlay anonymity networks. The most popular such service today is Tor [8], but others include JAP [19] (commercially offered as JonDonym [16]), i2p [17] and Anonymizer Universal (AU) [2].

All those designs employ relays to form a communication path between a client and a destination that hides information about who is connecting to whom from network observers, and from the destination itself. While they have been improved upon and have grown in popularity, anonymity networks remain notorious for being difficult to study [22, 37]. This is partly due to their inherent privacy properties, but also due to ethical considerations: they are live systems, and any data collection about their use may put in danger real users by compromising their anonymity.

Data collection systems, in this context, must be mindful of four main risks:

1. The network is run by volunteers and *anyone* with resources may join the network by contributing nodes with bandwidth or computation cycles to relay traffic. This limits the trustworthiness of nodes.
2. The data that may be collected at nodes is sensitive and directly publishing it may break the non-collusion assumption required by relay-based anonymity networks to maintain user anonymity.
3. The nodes that collect or process statistical information should not become targets of compulsion attacks by making them more attractive targets of miscreants and authorities.
4. Low-latency anonymity networks are vulnerable to correlation attacks [27, 28] that observe traffic volumes entering and leaving the network. Published statistics must hide information that would allow a client-side adversary with a partial view of the network (an ISP, for example) to mount such attacks.

To mitigate these risks, we propose PrivEx, a system for collecting aggregated anonymity network statistics in a privacy-preserving manner.

PrivEx collects aggregated statistics to provide insights about user behaviour trends by recording aggregate usage of the anonymity network. To further reduce the risk of inadvertent disclosures, it collects only information about destinations that appear in a list of known censored websites. The aggregate statistics are themselves collected and collated in a privacy-friendly manner using secure multiparty computation primitives, enhanced and tuned to resist a variety of compulsion attacks and compromises. Finally, the granularity of the statistics is reduced, through a noise addition method providing (ϵ,δ)-differential privacy, to foil correlation attacks.

The novel contributions in PrivEx are:

1. A safe mechanism to collect client statistics from anonymity network egress nodes;
2. Two secure multiparty protocols that protect the intermediate values of a distributed differential privacy (DDP) computation, optimized for the problem at hand;
3. Reduced noise in the results of the DDP computation leading to higher utility while still maintaining the desired level of privacy, both as tunable parameters;
4. A security analysis detailing the resistance to compulsion, compromise and correlation attacks; and
5. An evaluation of the overhead and performance of a proof-of-concept implementation of PrivEx.

There are three main motivations behind PrivEx. The first is that developers of anonymity networks have so far been unable to inspect egress trends. This information can guide designs that enhance performance and provide features that better address the needs of users. For example, network designers would like to be able to determine how much of the network's traffic is for the purpose of protecting the user's identity from the website she visits, and how much is for the purpose of censorship circumvention—protecting the identity of the website she visits from the *censor*. These different user bases have different security and privacy requirements, and knowledge of the prevalence of each set can help tune the network appropriately. The second motivation is to inform the research community with realistic information about us-

*This is an extended version of our CCS 2014 paper. [12] It was updated in July 2015 with results from a real-world deployment in §7.

age trends to guide research in censorship resistance mechanisms, performance tuning, and resource allocation, to name a few. Finally, one of the important open questions in any anonymity network is how to model client behaviour since this is exactly the information that needs to remain confidential. With realistic statistics we can shed light not only on client behaviour but also use it to ensure that when we test novel designs or system changes we can model their effects on clients in a more realistic manner, leading to more ecologically valid results.

Unfortunately, previous research on client behaviour [23] led to controversy due to private client information being gathered—even though it was destroyed and never exposed [34]. This set a precedent that client information, no matter how it is collected, is off-limits for legitimate research, which had a chilling effect on research in this area. Mindful of the risks to clients and respecting the fears of the privacy research community, PrivEx is a proposal that aims at resolving this deadlock by providing a means of safely collecting client traffic statistics in anonymity networks.

2. BACKGROUND

Anonymous Communication Networks.

Anonymous communication networks (ACNs) allow clients to hide their accesses to web pages and other Internet destinations from certain network observers (typically ones who can view network traffic on at most a small portion of the Internet).

Low-latency networks, such as Tor, JAP/JonDonym, or i2p obfuscate network traffic by routing it through multiple nodes: an *ingress* node, some number of *middle* nodes, and an *egress* node. The routing can be predetermined by the network, as in JAP/JonDonym, or source-routed subject to some constraints, as in Tor and i2p. To achieve security against network observers, traffic is encrypted so that the contents and metadata, including the destination, are only seen by the egress node and the client.

Simpler anonymizing networks, such as AU use only a single node and as a result are extremely susceptible to legal compulsion attacks (through court orders, for example) [30, 33]; hence, they will not feature in our discussions further.

Tor. Tor [8] is a popular ACN that provides anonymity by decoupling the routing information between the client and the destination. Clients use three intermediary nodes to route their traffic using onion routing. This prevents the destination from learning who the client is, and it also prevents an observer local to the client from learning which destination the client has connected to.

Tor, by default, uses three intermediate nodes in a connection between a client and destination (Figure 1). The client uses a telescoping mechanism to construct a *circuit* between herself and the last node, known as the exit node, which is the egress point of the client's traffic. As this is the location where PrivEx will perform its privacy-preserving data collection, we will refer to this node as the data collector (DC) in the remainder of the paper. Each client circuit has a unique identifier to help the DC manage the flow of traffic for multiple clients at once. The default behaviour is for the Tor client software to switch to a new circuit every 10 minutes.

The DC node knows the destination but not the originator of a connection. This is necessary to prevent it from relating the observed destination to any client and hence learn about her habits, activities or interests. Traditionally, exit nodes are required to delete any information about the connections that exit the Tor network through them. Publishing such information may be combined by an adversary (such as an ISP or national firewall) with a client-side view of the network to correlate exit activity with client activity to deanonymize the network traffic.

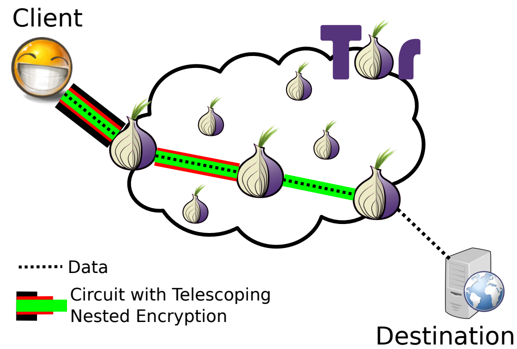

Figure 1: An overview of the Tor network and typical traffic flow (dotted line), highlighting Tor circuits, which use telescoping nested encryption.

Thus, to not introduce any new attack vectors, any effort to collect traffic data at exit nodes, even in aggregate, will have to minimize the information leaked to the adversary. This must hold even in the case that the adversary is able to compromise the node or compel the node operator to reveal the state of the node.

We will use Tor as a model ACN in which to integrate PrivEx in the discussions that follow. This should aid in clarifying the descriptions and to help the reader relate PrivEx to real-world ACNs, but does not restrict the generality and applicability of PrivEx to other systems.

Differential Privacy.

Traditional differential privacy [9] protects a central database—a table where rows hold sensitive data about individuals—that is to be kept private. This central database holds *raw* records that are only to be released to the public in noisy or aggregated form. The database allows multiple queries from clients who spend from a *privacy budget* for each query.

Established differential privacy mechanisms add noise to the results of client queries to ensure that personal information—*i.e.*, information about a particular entity that contributes to the results of a query—cannot be gleaned from those results. Intuitively, given any two "neighbouring" databases, one containing an entity's data and another without that entity's data, but otherwise equal, then the probability of observing any particular output to a given query will be close for the two databases.

PrivEx implements a privacy mechanism based on adding noise from a Gaussian distribution.[1] Adding an appropriate amount of Gaussian noise to the results of queries produces (ϵ, δ)-differential privacy: if D and D' are two neighbouring databases (as described above), then the probabilities $P_D(x)$ and $P_{D'}(x)$ that a given query outputs x when using the databases D and D' respectively, are related by $P_D(x) \leq e^{\epsilon} \cdot P_{D'}(x) + \delta$. [10].

In our setting, the database consists of one row for each censored website whose visits we wish to count, and queries will simply be of the form "output the counts in each row of the database (plus noise)".

3. THREAT MODEL

PrivEx maintains its security properties against an adversary that is local to the client or the website servers they visit. The adversary

[1]We discuss later why we use Gaussian instead of Laplacian noise.

is able to monitor traffic between the client and the ingress of the anonymity network, or traffic between the egress of the network and the client's destination, but not both at the same time. This assumption is similar to the one required to argue Tor is secure. As a result, this adversary is presumed to be unable to defeat the anonymity system. However, if any information is also revealed by the DC node, such as client usage statistics, that data could possibly be used to correlate traffic. A secure statistics gathering system, like PrivEx, should prevent any such attacks.

We allow the adversary to operate nodes in PrivEx; *i.e.*, deploy or compromise ingress nodes in the network and be part of the aggregation service itself. The adversary may also use the anonymity network to relay its own traffic in order to induce desired statistics into the aggregation process. Malicious nodes can report spurious data without generating or processing the corresponding traffic.

PrivEx is secure when there is at least one honest data collector and at least one honest-but-curious tally key server (described in §4). While dishonest data collectors can report "junk" statistics and malicious servers can disrupt the protocol, the security requirement in PrivEx is that no client traffic pattern information from honest data collectors is ever exposed: neither while it is stored on the data collectors, while it is in transit in the network, nor while it is being processed by the aggregating service. That is, malicious parties can disrupt the statistics reported by PrivEx, but cannot expose private data. In the distributed-decryption variant of PrivEx (see §4.2), we can further *detect* misbehaving servers. We discuss the security implications of malicious actors and publishing client statistics in further detail later in §5.1.

4. THE PrivEx SCHEMES

The goal of PrivEx is to count how many clients are visiting each of a list of particular known censored websites.[2] These statistics are gathered and reported in a privacy-sensitive manner so that the outputs of PrivEx cannot be used to perform traffic correlation attacks. Note that it is trivial to adapt PrivEx to collect statistics for any type of event that the egress nodes can count, such as the traffic volume per circuit, variance in circuit-management statistics, client navigation behaviour, and so on.

The DC nodes in PrivEx run on the same machines as the egress nodes of the underlying ACN. The DC listens for events of interest from the egress node, and securely aggregates them. In our setting, an event will consist of the ACN egress node reporting that a particular circuit has asked to perform a DNS lookup of a particular website.

PrivEx collects and aggregates statistics over a fixed period of time, called an *epoch*. We pick an epoch according to the granularity of the statistics we wish to collect—for our example ACN, Tor, we have chosen one hour as the epoch.

We introduce two PrivEx scheme variants that provide secure and private aggregate statistics of events collected by the DCs. They differ in the cryptographic primitives used to protect the data while it is in storage and in the protection that they offer against malicious actors.

The first scheme, based on secret sharing (PrivEx-S2), is secure in the honest-but-curious setting but can be disrupted by a misbehaving actor.

The second scheme, based on distributed decryption (PrivEx-D2), is secure in the covert adversary setting in that misbehaving servers can be identified. Most importantly, however, in both schemes, the disruption of the protocol by malicious parties does not result in information leakage.

[2]This list can optionally have an "Other" entry to count the *total* number of visits to non-censored websites as well.

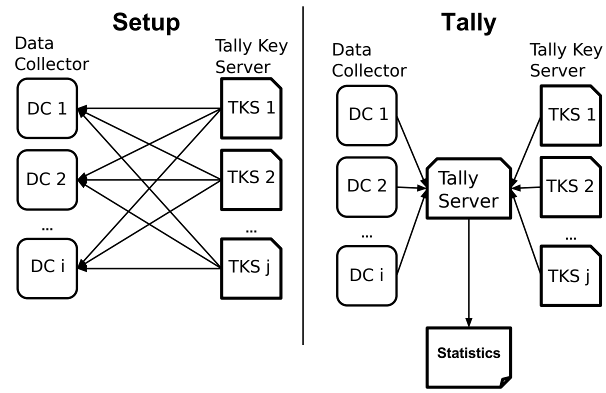

Figure 2: PrivEx variant based on secret sharing

4.1 PrivEx based on Secret Sharing

There are three types of participants in PrivEx-S2: Data Collectors (DCs), Tally Key Servers (TKSs), and a Tally Server (TS). The DCs relay traffic between the ACN and the destination; they collect the statistics we are interested in. TKSs are third parties who combine and store the secret shares received from DCs and relay aggregates of those secret shares to the TS. The TS simply adds up the secret shares provided by the DCs and the TKSs to produce the aggregated results. Figure 2 depicts an overview of our scheme.

Setup. At the beginning of every epoch, each pair of DC (i) and TKS (j) share a secret key (K_{ij}). This key can be the result of an ephemeral Diffie-Hellman exchange, or more simply, each DC i can seed each TKS j with a shared key through a secure channel (*e.g.*, TLS 1.2 using a ciphersuite that provides forward secrecy).

Each DC maintains a number of secure counters, each cryptographically storing the count of accesses to a specific destination (wID). The DC cryptographically initializes a database of records, each representing a secure counter, with the following schema: [wID, C_{wID}] where

$$C_{\text{wID}} = \left(n_{\text{wID}} - \sum_j \text{PRF}(K_{ij}; \text{wID}) \right) \bmod p$$

Here, n_{wID} is the noise for this counter (see §4.4), PRF is a keyed pseudorandom function, and p is a smallish prime (such as $p = 2^{31} - 1$). After this step, the DCs securely delete their shared keys K_{ij} and the noise n_{wID}.

Each TKS (j) also uses K_{ij} to compute its contribution to the count for each wID as:

$$S_{\text{wID}} = \left(\sum_i \text{PRF}(K_{i,j}; \text{wID}) \right) \bmod p$$

and then securely deletes its copy of the K_{ij}. Alternatively, in order to mitigate failing DCs, the TKSs can store the keys until the tally phase but this opens up the TKSs to compulsion attacks to reveal the keys, and hence the individual DC statistics.

Counting. Upon a DNS lookup event, the DC simply adds 1 to the appropriate secure counter as follows: $[\text{wID}, C'_{\text{wID}} = (C_{\text{wID}} + 1) \bmod p]$. We choose p large enough to expect no overflow of counting events—we can only reliably aggregate up to p events per counter.

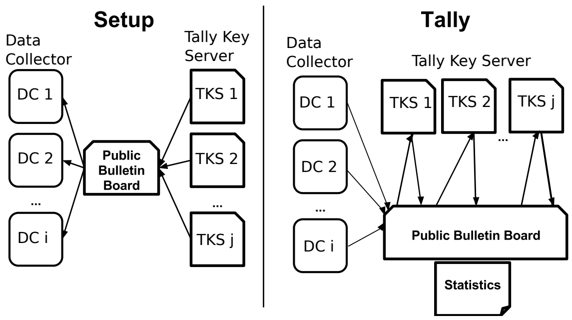

Figure 3: PrivEx, distributed decryption variant

Aggregation. At the end of every epoch, all the DCs and all the TKSs send their databases of secure counters to the TS.

The TS simply adds up all the shares received from the DCs and TKSs and publishes the results, which are the aggregated destination visit totals from all the DCs plus the total of the noise added by each DC at the setup stage in each counter. Once the results are published for the current epoch the tally server deletes the received data and awaits the next epoch's data to tally.

After sending their data for the epoch to the tally server, all the DCs and TKSs securely delete their databases and reinitialize through the setup phase, starting the cycle again.

4.2 PrivEx based on Distributed Decryption

We now describe PrivEx-D2, depicted in Figure 3. PrivEx-D2 utilizes the Benaloh encryption scheme [5]—a distributed additive homomorphic encryption scheme. This scheme is a variant on ElGamal: a (private,public) key pair is $(a, A = g^a)$ and an encryption of a message $m \in \mathbb{Z}_q$ with randomness $r \in \mathbb{Z}_q$ is $E_A(r;m) = (g^r, A^r \cdot h^m)$, where g and h are generators of a cryptographic group $\mathbb{G}$ of order q. Note the additive homomorphism: $E_A(r_1;m_1) \cdot E_A(r_2;m_2) = E_A(r_1 + r_2; m_1 + m_2)$, where the multiplication is componentwise. Decryption is $D_a(C_1, C_2) = DL_h(C_2/C_1^a)$. Note that decryption requires the taking of a discrete log, but if the message space M is small (as is the case for counts of website visits, or in Benaloh's original application, counts of votes), this can be done with the kangaroo [29] or baby-step-giant-step [31] methods in time $O(\sqrt{|M|})$, or even faster if more space is consumed by a pre-computation table.

Note that PrivEx-D2 uses a public bulletin board (PBB) instead of a Tally Server; the PBB is used as a repository of results and public keys from the DCs and TKSs. We can instantiate it with a database server which maintains tables for the TKS public keys and intermediate decryption results, and the final statistics of the epoch. To mitigate misbehaviour by an untrusted PBB, the messages stored thereon should be digitally signed by their authors using long-term authentication keys.

Setup. At the beginning of every epoch, each TKS (j) picks a random (ephemeral) private key $a_j \in \mathbb{Z}_q$ and computes its public key $A_j = g^{a_j}$. They publish the public keys to the PBB, along with a non-interactive zero-knowledge proof of knowledge (using the Fiat-Shamir heuristic) of the private key a_j. Each DC then checks each proof, and calculates the compound key A by taking the product of all the published keys: $A = \prod_j A_j$. Now each DC, for each secure counter for website w in its table, computes the amount of noise n_w to be added (see §4.4), and stores $E_A(r_w; n_w) = (g^{r_w}, A^{r_w} \cdot h^{n_w})$. Note that the randomness r_w will be freshly chosen for each counter, and discarded immediately after encryption, along with the plaintext n_w.

Counting. When the DC observes a visit to a website under observation, it multiplies (component wise) the appropriate encrypted counter by $E_A(r;1) = (g^r, A^r \cdot h)$ where r is random.[3] After c_w visits, the secure counter will hold $(g^r, A^r \cdot h^{c_w+n_w})$ for some r. It can optionally also re-randomize the all the other counters to ensure that two subsequent snapshots of the database do not reveal which counter has been incremented.

Aggregation. At the end of the epoch, each DC (i) publishes to the PBB a commitment to its encrypted counters for each website (w) under observation: $C\left(\langle (g_{i,w}^r, A^{r_{i,w}} \cdot h^{c_{i,w}+n_{i,w}})\rangle_w\right)$, where C is an appropriate commitment function. After all DCs have posted their commitments to the PBB, each posts the opening of its commitment (the list of encrypted counters $\langle(\alpha_{i,w}, \beta_{i,w})\rangle_w = \langle (g_{i,w}^r, A^{r_{i,w}} \cdot h^{c_{i,w}+n_{i,w}})\rangle_w$). Each TKS j then checks that the DCs' openings are consistent with their commitments, and consolidates the openings by computing $\alpha_w = \prod_i \alpha_{i,w}$. It then computes $\alpha_w^{(j)}$, TKS j's share of the decryption, as $\alpha_w^{(j)} = (\alpha_w)^{a_j}$, and posts that back to the PBB, along with a non-interactive zero-knowledge proof of equality of discrete logarithms to (g, A_j) to show that the computation was correct. Everyone can then check the proofs and compute the value $h^{c_w+n_w} = \left(\prod_i \beta_{i,w}\right) / \left(\prod_j \alpha_w^{(j)}\right)$, where $c_w = \sum_i c_{i,w}$ and $n_w = \sum_i n_{i,w}$. From here, $c_w + n_w$ can be computed using one of the discrete logarithm algorithms mentioned above. A proof of security for PrivEx-D2 can be found in the appendix.

4.2.1 Filtering Statistics by Client Origin

So far, we have assumed there is a single list of censored websites whose visits we are interested in counting. However, different websites are censored in different countries, and we may wish to count a visit to, say, Wikipedia if the user is in China, but not in the UK, a visit to the Pirate Bay if the user is in the UK, but not in Sweden, etc.

In this section, we present an extension to the PrivEx-D2 protocol that allows us to maintain *per-country* lists of censored websites, and only count a visit by an ACN user to a given website if that website appears on that user's country's list.

To do this, we of course need to determine what country the user is in. This is best done at the ingress point to the ACN, where the true IP address of the user is visible. Indeed, Tor already collects this information so that it can display per-country counts of numbers of users. [36] It is of course vital that the DC *not* learn this potentially identifying information about the client. The ingress node will therefore forward to the DC an *encrypted vector* encoding the country. The length of the vector is the number of countries N_C for which we are monitoring accesses to censored websites, plus one for "Other". The vector is then $V = \langle E_A(r_c; \delta_{c,c^*})\rangle_{c=0}^{N_C}$ where c^* is the country the user is in and δ_{c,c^*} is 1 if $c = c^*$ and 0 otherwise. The r_c are uniform random elements of $\mathbb{Z}_q$. The ingress node also provides a zero-knowledge proof that each element of the vector is an encryption of either 0 or 1, and that the sum of the plaintexts is 1. We note this is the same proof as used in electronic voting schemes, for example. [5]

The DC will check the zero-knowledge proof, and when it observes a connection to, say, Wikipedia, will multiply into its count not $E_A(r;1)$, as above, but rather $\prod_c V_c$, where the product is over those countries c that censor Wikipedia. The remainder of the pro-

[3]For a slight efficiency gain, r can be 0, so that the multiplication is by $(1, h)$. The downside is that this can leak information to an attacker that can observe the internal state of a DC at two different times within one epoch, yet cannot observe that DC's DNS lookups.

tocol is unchanged. Each vector V is associated to a circuit at circuit construction time and the DC knows which circuit requested the website.

4.3 PrivEx Scheme Comparison

Both schemes provide the security features we desire, but in some settings one may be preferable over the other.

In volunteer-resourced ACNs, such as Tor, some nodes will inevitably have low computation and bandwidth resources and it is best to minimize their computational, memory, and bandwidth overhead. In such cases, PrivEx-S2 is preferable since some messages are overall shorter and the computational overhead of frequent operations is smaller.

The length of the epoch can affect our choice of scheme since the relative time to set up and process the statistics increases for shorter epochs. While it is not a current requirement, if we wanted more near-real-time statistics, say every 5 seconds, then we would prefer PrivEx-S2 since the overhead is nearly negligible compared to PrivEx-D2. There are limits to how short the epoch can be, however, due to network latency affecting protocol communication.

On the other hand, PrivEx-D2 provides traitor detection of the TKSs and Denial of Service (DoS) resistance. In PrivEx-S2, any DC or TKS can DoS the system for the epoch if it does not report its statistics, whereas in PrivEx-D2 only DCs that report statistics for the epoch are included in the aggregation process and misbehaving TKSs (traitors) can be detected using cryptographic proofs ensuring that the computations were done correctly. Furthermore, PrivEx-D2 can optionally enjoy stronger perfect forward secrecy—against node seizure and adversaries that can view the memory contents multiple times in an epoch—by re-randomizing even those counters that have not been changed with every increment operation.

4.4 Calculating and Applying Noise

We introduce noise to our results to mitigate the risk of the correlation attack that reporting exact results may introduce. A more thorough discussion of the correlation issue is found in §5.2. In this section, we present the details of how the appropriate amount of noise is computed and added to the tallies.

4.4.1 How Much Noise?

We add noise to protect the privacy of users, but at the same time, if we add too much noise, it will hamper the *utility* of PrivEx; after all, we are deploying PrivEx to answer certain questions about ACN usage. We adopt a principled approach to adding noise to our statistics—one that allows the level of privacy and utility to be set to desired levels. For this purpose we have selected the model of differential privacy that can provide (ϵ, δ)-differential privacy through the addition of noise using a Gaussian mechanism with mean 0 and a standard deviation σ selected for the level of privacy and utility we require.

We wish to protect information about whether any individual user's information is in the published data set, or is not in it. To do this, we need to set an upper bound—called the sensitivity (S)—on the maximum contribution one user can make to the count in any epoch. For Tor, we use the fact that, by default, one circuit is created every ten minutes, so that if our epoch length is, say, one hour, and we always ignore repeated visits to the same website by the same circuit, we can set $S = 6$—the security implications of implementing this are discussed in §5.1.1. For other ACNs, an appropriate sensitivity can be similarly selected.

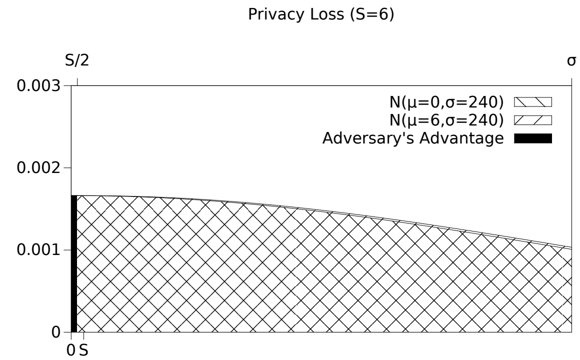

Figure 4: The advantage is 0.5% (shaded area) of the adversary in guessing the correct value of the statistic. Note the almost total overlap of the two probability distributions.

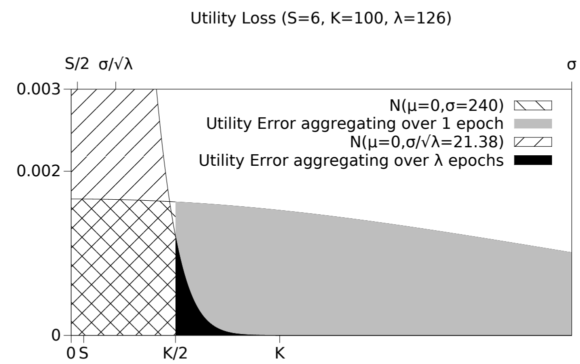

Figure 5: The probability of error is 0.1% (dark shaded area) when the reported statistic (averaged over $\lambda = 126$ epochs) appears closer to K than to 0. Compare this to the much larger error of 41.75% (lighter shaded area) when $\lambda = 1$.

As we are interested in practical applications of PrivEx, we provide the means to calculate the exact values of ϵ and δ through the lens of the privacy and utility levels we desire.

What we are interested in controlling is the *advantage* (over random guessing) of an adversary in guessing whether a particular user's data is contained in the published (noisy) statistics, *even if the adversary knows all the other inputs to the statistics*. That is, discounting the known information, the adversary is trying to determine whether the published statistics are more likely to represent a true measurement of 0 (the user is not present) or S (the user is present).

Therefore, the adversary's task is to tell if a given statistic is drawn from the distribution $N(0, \sigma)$ or $N(S, \sigma)$. Given a reported statistic, if it is less then $\frac{S}{2}$, the adversary's best guess is that the true statistic is 0, and S otherwise. It is easy to see that the advantage of the adversary is then given by the area under the $N(0, \sigma)$ normal curve between 0 and $\frac{S}{2}$, as depicted in Figure 4.

The adversary's advantage can then be minimized by selecting σ large enough such that $Pr[0 < N(0, \sigma) < \frac{S}{2}] = Pr[0 < N(0, 1) < \frac{S}{2\sigma}]$ is as close to 0 as desired. However, choosing σ too large will hamper utility, as we discuss next.

To address our utility needs, we must first decide on a question to ask. A typical question would be, "On average, how many visits are there to a given censored website per epoch?", and we may be content to know the answer to within some *resolution* K, say 100 or 1000. This gives us two benefits over the privacy adversary: we only care about average behaviour over many epochs, and not specific users at specific times (in order to carry out a correlation attack); and we only care about results to within K, not to within single users' contributions.

If we average over λ epochs, the standard deviation of our noise becomes $\frac{\sigma}{\sqrt{\lambda}}$. Then, if we want to distinguish two hypotheses that differ by K (*e.g.*, does this website see closer to 0 visits per epoch or closer to $K = 1000$ visits per epoch over the ACN—a question we cannot answer today), our *utility error*—the probability we answer incorrectly—is $Pr[N(0, \frac{\sigma}{\sqrt{\lambda}}) > \frac{K}{2}] = Pr[N(0,1) > \frac{K\sqrt{\lambda}}{2\sigma}]$, as depicted in Figure 5. Slightly different questions would produce slightly different formulas for the utility error, but they will be computable in the same vein.

Therefore, for a given sensitivity S and tolerance P on the advantage of the privacy adversary, we can work out the desired standard deviation σ for our noise by solving for $Pr[0 < N(0,1) < \frac{S}{2\sigma}] \leq P$ using a standard normal curve z-table. Then, given a tolerance U on the utility error, and a resolution K for our question, we can determine the number of epochs λ we will need to average over by solving for $Pr[N(0,1) > \frac{K\sqrt{\lambda}}{2\sigma}] \leq U$ similarly.

In the presence of possibly malicious DCs, the situation is only slightly more complicated. Malicious DCs (who do not immediately forget the amounts of noise with which they initialized the secure counters) know the amount of noise they added. By removing that from the reported tally, the remaining amount of noise (contributed by the honest DCs) is less than expected.

As we will see in §4.4.2, each DC i adds noise selected from a normal distribution whose standard deviation is proportional to its *weight*—the probability w_i that that DC will be selected by a user. If we can assume a lower bound H on the total weight of honest DCs, we can adjust the above calculations in a simple manner. (In §5.1.2 we will argue that $H = 0.8$ is a reasonable lower bound for Tor.) Honest DCs tune the amount of noise to add by adjusting the value of σ to $\sigma_H = \frac{\sigma}{H}$. This has the effect that honest DCs add more noise so that it maintains the desired privacy level, at the expense of requiring an increase in λ by a factor of H^{-2} (an increase of about 56% for $H = 0.8$) to achieve the same level of utility as before.

A Worked Example. Using Tor as our ACN, and one-hour epochs, so $S = 6$, we want to find σ given a desired privacy adversary advantage of at most 0.005. Consulting a z-table, we find that we want $\frac{S}{2\sigma} \leq 0.0125$, so $\sigma \geq 240$. Then, if we want utility error $U = 0.01$, the z-table says we need $\frac{K\sqrt{\lambda}}{2\sigma} \geq 2.33$, so for $\sigma = 240$, $K\sqrt{\lambda} \geq= 1120$ will suffice. Then if $K = 1000$, λ can be as low as 2 epochs, if $K = 100$, then $\lambda = 126$ epochs (or 5.25 days), but to get an average number of visits per epoch to within $K = 1$, we would need over 140 *years*.

We now analyze the case where some fraction of DCs may be malicious. Assume that we expect that the total honest weight is at least 80%. We adjust σ to $\sigma_H = \frac{\sigma}{H} = 240/0.8 = 300$. Then, for the same utility error as above, $K\sqrt{\lambda} \geq 1400$ will suffice. For the same values of K we would now need 2 epochs, 8.2 days, and over 224 years respectively.

In the preceding analysis we only need consider the amount of noise to add in terms of the standard deviation σ of the distribution we sample from. We can link this back to (ϵ, δ)-differential privacy by observing the parameters' relation to σ as follows [14]:

$$\sigma = \frac{S}{\epsilon} \cdot \sqrt{\ln\left(\frac{1.25}{\delta}\right)}$$

Thus, rather than, as in previous works [10, 14], having the system designer select not-very-meaningful values of ϵ and δ, and computing σ as above to determine how much noise to add, we *instead* determine σ directly using parameters specifically pertinent to the system and to the questions it is trying to answer.[4]

4.4.2 Distributed Noise Application

The DCs independently apply the noise as we never want the raw (un-noisy) data to be divulged. We can distribute the application of noise since we know from Dwork *et al.* [11] that if individual databases are differentially private then so is their sum.

A naive way to go about this, and one that avoids the use of third parties, is for the DCs to publish their noisy data directly to the public. The consequence of this is that each DC would need to add enough noise so that its *individual* statistics provided the desired bound on the advantage of the privacy adversary. This would make the total noise considerably larger (by a factor of the square root of the number of DCs), and so the number of periods λ to average over must increase by a factor of the number of DCs in order to keep the desired bound on the utility error.

This is why PrivEx works with *global* noise instead of *local* noise: each DC adds some amount of noise, whose *total* is distributed as $N(0, \sigma)$ for the desired σ, but does so using secure multiparty computation so that the individual noise components are never revealed.

We then need to calculate how much noise *each* DC should add. What we want is for each DC i to add noise from $N(0, \sigma_i)$, where σ_i is proportional to the probability w_i that the DC will get used. In Tor, for example, high-bandwidth nodes get used with higher probability, so they will see more usage, and add more noise, while more impoverished nodes will have less usage and less noise.

Then, given the desired σ, we want to solve for the σ_i such that $\sigma_i \propto w_i$ (so $\sigma_i = w_i \cdot \phi$ for some ϕ independent of i) and $\sum_i N(0, \sigma_i) \sim N(0, \sigma)$. Since $\sum_i N(0, \sigma_i) \sim N(0, \sqrt{\sum_i \sigma_i^2})$, we have that $\sigma^2 = \sum_i \left((w_i \cdot \phi)^2\right)$, so solving for ϕ, we find that $\sigma_i = w_i \cdot \phi = \sigma \cdot \frac{w_i}{\sqrt{\sum_i (w_i^2)}}$. In PrivEx, the values of ϕ and σ are made available to the DCs from the PBB or TKSs.

That we are adding together a potentially large number of independent noise sources is the reason we target Gaussian rather than Laplacian noise: while adding many Gaussians yields a Gaussian, a Laplacian distribution cannot be decomposed into sums of other independent random variables.

We note that, when adding noise, it is important for each DC to preserve non-integral and negative values for the noisy count, so that, when added together, extra biases are not introduced. As the encryption used in our counters takes integer plaintexts, we must use a fixed-point representation where all of our values are expressed as multiples of some small number γ. If there are N DCs, then in order that adding N values of resolution γ together will be unlikely to produce an error of more than 1, we set $\gamma \leq \frac{1}{2\sqrt{N}}$.

For $N \approx 1000$, as in the current Tor network, $\gamma = 0.01$ will suffice.[5] Note, however, that this fixed-point representation expands the plaintext space by a factor of $\frac{1}{\gamma}$, and so increases the time to compute the discrete logarithm in the final step of the PrivEx-D2 protocol by a factor of $\frac{1}{\sqrt{\gamma}}$.

4.4.3 Targeted Temporal Queries

[4]Since we only ever make one query we do not need to calculate how much privacy budget we have left after publishing our aggregated statistics.

[5]This also deals with an issue with rounding and differential privacy identified by Mironov. [25]

PrivEx publishes the noisy total statistics for each epoch. The amount of noise is computed to protect privacy, and a number of epochs' statistics must be averaged to gain utility. However, these epochs do not need to be consecutive, so, for example, one could ask questions like, "Is Wikipedia visited via this ACN more often on weekends or weekdays?". The *number* of epochs to average will not change, however, so if the epochs of interest are spread out in time, the total time to answer such a question will increase.

5. SECURITY ANALYSIS

5.1 Resistance to Attacks

We now address the attacks that are of the most concern. Recall that our requirement for security is that PrivEx should not reveal private information to an adversary, even if it fails to produce meaningful answers to the system designers' questions.

5.1.1 Legal or Other Compulsion

A DC can be compelled to reveal its database of collected statistics through a legal order or extra-legal compulsion. If this database is stored in the clear then privacy would be violated. PrivEx mitigates this threat by storing an encrypted database with the property that the DC cannot decrypt the database on its own. Recall that at the setup stage in PrivEx, all DC databases were encrypted using shared keys with, or public keys of, the tally key servers.

The adversary can also compel the servers to comply in the decryption of individual DCs' measurements (with less noise than the aggregate). This would indeed be troublesome, but we mitigate this by ensuring that the PrivEx servers are distributed across diverse legal boundaries making compulsion infeasible. Indeed, as long as at least one server is uncompromised then all DC data is safe. Furthermore, since we start with fresh keys for each epoch, this compulsion could not occur retroactively.

PrivEx requires that we bound the sensitivity—the maximum number of times one client can access a particular website in one epoch. We do this by maintaining, in plaintext, a list of websites visited during the *lifetime* of a circuit, which is 10 minutes in Tor. This introduces a potential information leak if the adversary is able to compromise an honest DC *while* circuits are being served; this would reveal the censored websites visited by each circuit. While this in itself does not link a client to a destination an adversary may use this information to correlate traffic patterns it can record at the client side of the circuit. However, if the adversary can compromise an ACN relay while it is actively serving an open circuit, then the encryption keys it could recover could compromise those circuits anyway even without access to the plaintext list.

5.1.2 Malicious Actors

Data Collector. The DC can behave maliciously by reporting untrue statistics. While there is no safeguard to an attack on the integrity of the statistics we are interested in, the confidentiality of the statistics collected at other DCs and the aggregate statistics that are output by PrivEx are safe from the actions of a misbehaving DC as long as the security of the encryption schemes that we use remains intact. We may mitigate the impact of this attack by using range proofs at additional computation and communication costs, but this still does not remove the threat entirely. In §4.4.1 we suggested that $H = .8$ is a reasonable lower bound on the amount of honest DC weight for Tor. The reason we give this value is that if more than 20% of the exit weight of Tor is compromised, then Tor is sufficiently susceptible to circuit linking attacks [1], and could more easily compromise clients without using the less-noisy statistics provided by the degraded PrivEx.

Finally, we note that if a DC is compromised, the adversary can also perform a correlation attack, and can likely read the memory, including encryption keys protecting any active circuits, thus retroactively deanonymizing them. This is a shortcoming of the underlying ACN; PrivEx does not exacerbate this problem.

Tally Key Server. The tally key servers collectively play a critical role in the PrivEx schemes and hence are vectors of attack. A bad actor may try to gain access to the statistics in a less secure manner or an insecure intermediate form (*i.e.* without noise).

We guard against this in both variants of PrivEx by ensuring that in the setup stage all DCs initialize their databases by encrypting each secure counter using the key material provided by, or shared with, all the participating TKS servers. This ensures that even if all but one TKS try to decrypt the data in an information-leaking manner, a single honest server's key material and noise added by the DCs prevents any information from being revealed.

In PrivEx-S2, a single DC or TKS can launch a denial of service attack by not sending its share, which would mean that for that epoch no results could be determined. In PrivEx-D2, we can identify the misbehaving TKS, which introduces consequences to DoSing. In either case, no private information is leaked.

Tally Server and Public Bulletin Board. The TS and PBB are unable to learn anything extra by misbehaving since none of the intermediate data is ever in the clear and their inputs and outputs are public, making verification possible.

5.2 Correlation Attack with Auxiliary Information

Data Collector traffic information may not reveal anything on its own, but there is a danger that an attacker could fruitfully combine it with auxiliary information, such as observations of a target user's, or indeed of many users', network traffic.

For example, if we did not add noise, but simply released accurate counts only if they were in excess of some threshold, then an adversary could generate its own network traffic to push the counts above the threshold, and then subtract its own traffic (for which it knows the true counts) to yield accurate counts of potentially a single user's traffic.

The differential privacy mechanism proposed adequately addresses this threat. It ensures that, for any adversary, the response of PrivEx if the target user *did* visit a target website in a given epoch will be hard to distinguish from the response if the user *did not*.

We also note that since there is only one question PrivEx answers (How many visits were made via the ACN to each of this list of websites in this epoch?), and it can be asked only once per epoch, differential privacy's notion of a "privacy budget" is not required in our setting.

6. IMPLEMENTATION

We have built proof-of-concept implementations for both variants of PrivEx. They are implemented in Python using the Twisted library for asynchronous networking between the parties of the system.

Each PrivEx scheme uses a few hundred lines of python code. The code is available for download from our PrivEx website.[6] Both schemes use TLS 1.2 with ECDHE for communication between endpoints to ensure that the key material remains confidential during transit and benefits from perfect forward secrecy. We set up long-lived TLS connections when PrivEx first comes online; their

[6] `https://crysp.uwaterloo.ca/software/privex/`

Operation	Total per epoch (ms)	Per node (ms)
TKS initialize	0.012 ± 0.004	0.0012 ± 0.0004
TKS register	41000 ± 3000	4100 ± 300
DC initialize	40000 ± 3000	40 ± 3
DC register	312 ± 8	0.312 ± 0.008
DC increment	900 ± 90	0.90 ± 0.09
DC publish	1.7 ± 0.1	0.0017 ± 0.0001
TKS publish	0.56 ± 0.06	0.056 ± 0.006
TS sum	470 ± 20	470 ± 20
Epoch Total	83000 ± 6000	—

Table 1: The overhead per epoch (with 95% confidence intervals) incurred by participants in the PrivEx-S2 scheme for 10 TKSs and 1000 DCs with 1000 websites with one million visits per epoch.

Operation	Total per epoch (ms)	Per node (ms)
TKS initialize	10.9 ± 0.2	1.09 ± 0.02
DC combine key	4.05 ± 0.02	0.00405 ± 0.00002
DC initialize	295000 ± 600	295 ± 0.6
DC increment	3.9 ± 0.1	0.0039 ± 0.0001
PBB productize	50400 ± 400	50400 ± 400
TKS decrypt	448000 ± 3000	44800 ± 300
PBB DL Lookup	6293 ± 40	6290 ± 40
Epoch Total	800000 ± 3000	—

Table 2: The overhead per epoch (with 95% confidence interval) incurred by participants in the PrivEx-D2 scheme for 10 TKSs and 1000 DCs with 1000 websites with one million visits per epoch.

communication and computational costs are amortized over many epochs.

We have not implemented the Country of Origin feature at this time since we would like to see PrivEx deployed in the Tor network with the core feature set before expanding on it. The core implementation above is ACN agnostic, and we aim to integrate it with Tor in the near future.

In the tables in this section, the "Per node" column is calculated by taking the total cost for each type of PrivEx node and dividing it by the count of that type of node, to find the cost at each type of node. This helps identify potential bottlenecks.

Computational Overhead.

We present PrivEx overhead statistics to show that both schemes have low computation requirements. The hardware for our experiments is a 3 GHz quad-core AMD desktop computer with 4 GiB of RAM running stock Ubuntu 14.04.

Using a test harness we measure the time the core components of each PrivEx scheme take under parameters typically found in the Tor network. We simulate a network of 10 TKSs and 1000 DCs; the latter reflects the number of exits in the current Tor network. The number of censored websites to collect statistics for is 1000 and each website is "visited" one million times. No actual website connections are made by the DCs since we are interested in capturing the overhead of the PrivEx schemes.

PrivEx-S2. From Table 1, we note that the setup phase of PrivEx-S2 takes 4.1 s on average and that the tally phase takes 470 ms on average (adding the "per node" times, as the nodes act in parallel). Without any ACN traffic (*i.e.* no DC increment operations), the total overhead wall-clock time per epoch is 4.6 s. The key figure to note is that the addition operations at the DC nodes take less then 1 μs each (900 μs for 1000 visits per DC) on average. This low cost is important, as this operation will be called the most often and the impact on DC nodes must be negligible so that they can service ACN requests without undue overhead.

PrivEx-D2. From Table 2, we note that the setup phase of PrivEx-D2 takes 297 ms on average with the DC nodes bearing the most cost. The entire tally phase takes 1.69 m on average per epoch (adding the "per node" numbers, as these operations occur in parallel). Combining the overhead for both phases, the epoch overhead wall-clock time is 1.7 m on average. We see in PrivEx-D2 that the addition operation takes 3.9 μs on average and again, like PrivEx-S2 above, this is desirable since it is the most frequent operation.

Discussion. PrivEx-S2 has lower computational cost than PrivEx-D2, by a factor of almost 10 in our example. Yet, it is clear from these results that the computational overhead at each type of node in PrivEx is low and that the time requirements are a small fraction of the duration of an epoch. Indeed, even if there are applications where statistics need to be gathered for shorter epochs, PrivEx can still be useful; as we saw earlier, for each setup-tally cycle the PrivEx-S2 scheme incurs less than 4.6 s of overhead while the PrivEx-D2 scheme incurs less than 1.7 m of overhead, meaning the statistics collection frequency can be as low as 5 s and 2 m respectively. This flexibility allows one to match the appropriate PrivEx scheme to the application's statistics frequency and threat model requirements.

Communication Overhead.

We now give a closed-form analysis of the communication costs of the two PrivEx schemes. In the following description, DC_N, TKS_N , and W_N represent the number of DC nodes, TKS nodes, and websites for which we are collecting statistics respectively.

An overhead in common for both schemes is the list of websites and the constants for DDP calculations σ, ϕ, and γ. We make the conservative assumption that the website domain name in the URL will not be more than 255 characters long, therefore the maximum length of the URL list is $255 \cdot W_N$ bytes. The constants require 8 bytes in total. In the experimental setting above this overhead is ~249 KiB, the overwhelming majority of it being the website list. While it is not as significant, we note that the website lists and values for the constants need not be transmitted every epoch, instead only being sent when there is a drastic change in the network conditions or the website lists are updated.

PrivEx-S2. In the setup phase, each DC sends 16 bytes of key material to each TKS for a total of $16DC_N \cdot TKS_N$ bytes.

In the tally phase, each DC sends 4 bytes to the TS for each website in the database for a total of $4W_N \cdot DC_N$ bytes. Similarly, each TKS also sends the same amount to the TS for each website for a total of $4W_N \cdot TKS_N$ bytes.

In each epoch, the total communication cost, in bytes, is

$$16DC_N \cdot TKS_N + 4W_N(DC_N + TKS_N)$$

For 10 TKSs and 1000 DCs tracking 1000 websites we see from Table 3 that the total communication cost for every epoch is ~4 MiB, but the cost for each type of node is far lower at only ~4 KiB.

PrivEx-D2. In the setup phase, each TKS sends 96 bytes of key material and zero-knowledge proof to the PBB for a total of $96TKS_N$ bytes. Then, each DC retrieves the key material and the proofs from the PBB for a total of $96TKS_N \cdot DC_N$ bytes.

In the tally phase, each DC sends a 32-byte commitment to the PBB for a total of $32DC_N$ bytes. After all DCs have sent their

	Setup	Tally	Total	Per node
DC	156.25	3906.25	4062.50	4.06
TKS	0	39.07	39.07	3.91
TS	0	0	0	0
Total	156.25	3945.32	4101.56	—

Table 3: Communication cost (in KiB) of PrivEx-S2 for 1000 websites, 10 TKSs and 1000 DCs per epoch, using closed-form analysis.

	Setup (KiB)	Tally (MiB)	Total (MiB)	Per node (KiB)
DC	0	61.07	61.07	62.54
TKS	0.94	0.31	0.31	31.74
PBB	937.5	641.17	642.09	657500
Total	938.44	702.55	703.47	—

Table 4: Communication cost of PrivEx-D2 for 1000 websites, 10 TKSs and 1000 DCs per epoch, using closed-form analysis. Note the units in the column headings.

commitments, the PBB sends each DC the commitments of the other DCs for a total of $32DC_N^2$ bytes. Then, each DC sends a 64-byte opening of the commitment for each website to the PBB for a total of $64W_N \cdot DC_N$ bytes. The PBB then sends, in parallel, the opening of the DC's commitments to each TKS for a total of $TKS_N(DC_N(64W_N + 32))$ bytes. In response each TKS sends the results of the partial decryption for each website in the database, along with a zero-knowledge proof of equality of discrete logs for a total of $TKS_N(32W_N + 64)$ bytes.

In each epoch, the total communication cost, in bytes, is

$$32\,\big(W_N(2D_N \cdot TKS_N + 2D_N + TKS_N)$$
$$+ D_N^2 + 4D_N \cdot TKS_N + 5TKS_N + D_N\big)$$

From Table 4 we see that, in our experimental setting, the total communication cost for each epoch is ~703 MiB, while each of the TKS and DC nodes send only ~32 KiB and ~63 KiB respectively. The bulk of the communication cost is borne by the PBB node.

Discussion. Both schemes scale linearly with the number of websites and TKSs. PrivEx-D2 scales quadratically with the number of DCs while PrivEx-S2 remains linear. While it is true that the PrivEx-D2 scheme is generally more expensive, we note that each DC and TKS transmits only tens of KiB of traffic per epoch, which is comparable to PrivEx-S2. However, the PBB transmits hundreds of mebibytes due to the higher security and privacy guarentees it allows. To mitigate the impact of this load, it is expected that the PBB will be well resourced for this task. Indeed, we expect that in real deployments the number of TKSs would be closer to three and the number of websites would be closer to 100. In that scenario, the total communication cost would be approximately 55 MiB per epoch.

The PrivEx-S2 scheme is relatively lightweight, enjoying very low overhead and perhaps a better choice in low-bandwidth environments or where the size of the website list will be very large.

Even so, in absolute terms, both PrivEx schemes have low overhead for DC and TKS nodes. We note that in the Tor network, even relays in the 1^{st} percentile by bandwidth (18.4 KBps)—which are also the least likely to be chosen in circuits in any event—can manage the load easily. [35]

From the perspective of the DC, which is also a node in the ACN, PrivEx does not significantly impact bandwidth usage which can be better used to service ACN traffic. From the perspective of the TKS, TS, and PBB, even though we expect that the servers would be well provisioned for the task of aggregating statistics, the resource requirements are low enough that they would also not be significantly impacted by participating in PrivEx.

7. REAL-WORLD DEPLOYMENT

Having designed and implemented PrivEx, we now use it to learn about the nature of censorship traffic on the Internet. We are particularly interested in seeing the breakdown of censorship traffic as it compares to non-censorship traffic on a particular network.

Methodology.

For our study we target the Tor network for two reasons. First, a privacy-preserving study of this nature has not been conducted and would yield useful insights about Tor user behavior, specifically how much traffic is censorship resistance related. Second, it would provide a proof-of-concept validation to the community that privacy-preserving and utility-preserving data collection is practical and spur PrivEx uptake and further research of these types of systems.

The first reason above may seem counterintuitive since we have classified Tor as a CRS, and hence all traffic on the network should be considered censorship resistance related. From an abstract and global Internet perspective this observation is certainly true. However, the Tor network is an ecosystem serving many purposes, including but not limited to censorship resistance. For a censor to block Tor—where such a block would have potential collateral damage due to the defensive strategies employed by Tor through their pluggable transports framework—the cost of information leakage due to the CRS activity on the network must be higher than the cost of the collateral damage it would suffer. Thus, knowing the base rate of CRS activity helps fill in information that would help evaluate if a block is economically responsible. Indeed, this analysis can tell Tor designers if more collateral damage needs to be leveraged to tip the balance to prevent such a block. Hence, Tor is an appropriate candidate for this type of study.

We utilize DNS requests as a means of learning about how often Tor users are interested in CRS-related websites. We are aware that a DNS request does not necessarily translate to an actual visit; *e.g.*, web pages that track users through third-party advertising networks will cause DNS queries to third-party domains but the user never actually "visits" those domains. This is acceptable for our study since we assume that a domain appearing on the censor's blacklist is due to it being a target of censorship.

We compiled a list of censored websites by scraping the GreatFire.org website that tracks Chinese censorship by running connectivity tests from behind the national firewall to websites on the Internet.[7] We only considered websites that were confirmed to be blocked. We augmented this list with a leaked list of websites blocked in Germany.[8] Altogether our compiled list had around 6100 entries.

Apparatus.

We utilized the PrivEx-S2 variant for two main reasons. First, neither CRS-client nor CRS-server software would have to be modified. Second, the low operational resource requirements lowered

[7] https://en.greatfire.org/

[8] https://bpjmleak.neocities.org—see archived version at https://web.archive.org/web/20140707204711/https://bpjmleak.neocities.org/ for the list that has since been removed due to pressure from the German authorities.

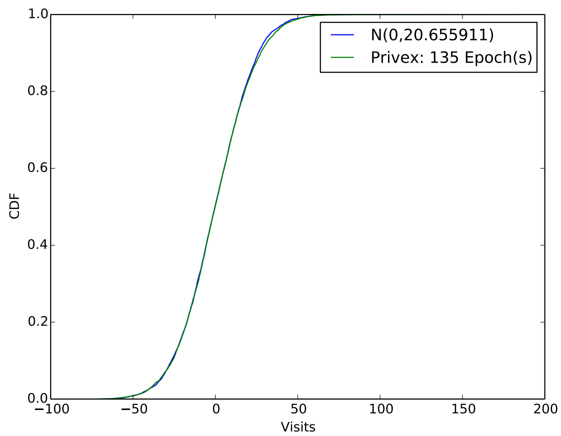

Figure 6: A CDF of the aggregated statistics collected by PrivEx as compared to the Gaussian noise added. A follow-up Kolmogorov-Smirnov test confirms that actual visits were registered in the statistics collected with PrivEx.

the bar for entry and were helpful in recruiting volunteers and resources.

Our initial proof-of-concept deployment consists of two TKSs, a TS, and a DC. The TKSs are operated by third parties, one using a virtual hosting provider on the Isle of Man and the other through the university network provider at the KU Leuven in Belgium. The DC is co-hosted with the Tor exit node, nicknamed gurgle, operating at the University of Waterloo in Canada. The TS is hosted on another machine at the same institution.

For the duration of the data collection reported here, gurgle had a probability of 0.15% of being selected as the egress node from the Tor network. This means that we expect to see this proportion of all traffic exiting the Tor network.

Collected Results.

We set the epoch to one hour and collected statistics for 135 epochs, which is more than the 126 epochs required for the level of privacy and utility from our worked example in §4.4.1.

We validate that our implementation produces results with the same characteristics as our analysis indicates and that they are reliable. We produce a CDF (Figure 6) of the aggregated statistics and plot it against a CDF of the Gaussian noise function we utilized with the standard deviation set to $\frac{\sigma}{\sqrt{epochs}} = \frac{240}{\sqrt{135}} = 20.655911$, where σ is the same as in the worked example above. We then ran a Kolmogorov-Smirnov test to ensure that the distance between the two plots was positive and large enough to indicate that they were drawn from two different distributions. The test showed a distance of 0.016 with likelihood of between 0.05 and 0.10 that the observed difference is due to randomness. This indicates that there were actual visits to the domains in our list and that we applied the expected level of noise.

The results show that the average number of hits from the censored list is the range 586–686 and those for off-list DNS requests is the range 31810–31910. This is a likelihood in the range of 1.8–2.2% that a given DNS request coming to gurgle is for a site in our compiled list, with probability exceeding 99%. This result provides an idea of the magnitude of the answer to the question of how CRS-related traffic compares to the rest of the network.

From this measure we draw a conclusion that since the base rate is so low, the accuracy of the censorship apparatus must be of a higher magnitude in order to avoid a large amount of false positives, *i.e.* collateral damage. For example, using the higher base rate of 2.2% above, a censorship apparatus that provides a 100% true positive rate (*i.e.*, no information leaks) and a 1% false positive rate (*i.e.*, collateral damage) would only be correct 69% of the time when it claimed that a some event was CRS related. The rest is collateral damage, in stark contrast to the ostensible 1% rate stated above. To compensate for the additional error and achieve the original 1% figure the apparatus would need to have a false postive rate of 0.01%. If a censor wants to ensure negligible collateral damage in this low base rate setting their apparatus must have a false positive rate of 0.001% which may be very difficult to achieve.

8. RELATED WORK

Differential Privacy. While PrivEx utilizes differential privacy (DP), there are many key differences in the setting in which it is traditionally applied and the PrivEx setting.

In classical DP there is a trusted centralized database—who is usually a third party host—which can see the real data and is considered secure. Instead, in PrivEx the data is distributed across nodes in the network where no entity has access to all of the real data from all of the nodes. The only data that is revealed to anyone is the aggregated statistics with noise added. An adversary would have to compromise a large fraction of the DCs, or *all* of the TKSs, in order to access the private data of the honest parties.

In the usual DP setting the database is static across epochs and clients use up their *privacy budget* to make a number of database queries—the results of which are usually private unless they choose to make them public. As discussed at the end of §5.2, in PrivEx, the database is completely refreshed at the start of every epoch and only a single constant query is ever made every epoch, the result of which is then made public.

A number of works consider the problem of securely computing functions in a distributed differential privacy setting.

Dwork *et al.* [10] provide a method for generating shares of random Gaussian noise in a multiparty setting mirroring the distribution of noise in our setting. The key difference is that the parties work together to first produce noise shares which are then used to perturb the data in their individual databases where as in PrivEx the noise is calculated independently using network state and does not incur extra protocol rounds. Also, they assume that $\frac{2}{3}$ of the participants will be honest while PrivEx makes no such explicit restriction, *i.e.* a lone honest DC may enjoy the same level of privacy as the designer intended, albeit with longer aggregation periods to gain the same level of utility as designed.

In the two-party setting of distributed differential privacy, Goyal *et al.* [13] explicitly evaluate the accuracy-privacy tradeoffs for computing Boolean functions. Mironov *et al.* [26] investigate calculating the distance between two vectors while McGregor *et al.* [24] do the same for Hamming distance. All these works explore the limits of DDP in the two-party setting. We contrast our work by noting that we consider a different type of problem (the summation of integral inputs) and we evaluate the tradeoff between the accuracy and privacy in the multiparty setting.

The closest related work is by Beimel *et al.* [4]. The inputs in that setting are binary, while those in ours are integral. While the binary inputs can indeed be adapted to integers, there remain three key differences. Their protocol requires more rounds of communication than ours, while we also allow for malicious parties, making PrivEx a more practical solution in our setting. Finally, in their setting, to preserve DP, the database of each DC is kept private and only binary outputs are released, whereas in our setting all DCs release their private data, albeit with noise added to preserve DP.

Also of interest is work by Kasiviswanathan *et al.* [18] where network graphs are analyzed to investigate how the removal and addition of nodes in the graph affect the privacy of the information about the structure of the graph. While they also consider differential privacy in the network setting, the key difference is that they investigate ways to safely reveal information about the nodes of the network themselves, whereas we are interested in the information that can be revealed by studying the traffic flowing *through* the network; *i.e.*, the network users' information.

A general key difference to the previous literature is that PrivEx provides a way to reason about the privacy and utility that the system provides whereas these previous works leave it up to the system designer to work out. We provide an explicit statement of, and relationship between, privacy and utility that are pertinent to data collection in ACNs—this provides an easier-to-analyze system and potentially an easier path to deployment.

Secure Multiparty Computation. Secure multiparty computations have been used in scenarios where the parties that perform the operations are not trustworthy. This means that they should not learn the inputs of the calculations, should provide (implicit or explicit) proofs that the calculations were performed correctly, and should not learn anything more than the output of the calculation.

A closely related work is SEPIA [6] by Burkhart *et al.* where networks collect data and wish to learn aggregate information about their networks without revealing their individual inputs. It develops a number of operations that can be performed on network data that can be evaluated by a pool of servers in a secure multiparty computation. While both PrivEx and SEPIA try to achieve similar goals in the collection of network statistics and use similar secret sharing schemes, there are a number of differences. First, while the authors of SEPIA briefly mention differential privacy as a possible defence, PrivEx provides a thorough treatment of how to use differential privacy to protect the aggregated statistics in a principled manner. Related to that is that SEPIA also requires that honest DCs sanitize their inputs, *i.e.* remove sensitive information, whereas PrivEx accomplishes the same with the addition of DP-noise. Second, PrivEx is secure as long as there is one honest data collector—adding the appropriate level of noise, as outlined in §4.4.1—and one honest TKS. This is in contrast to the SEPIA requirement that at least half of the aggregators be honest. This is especially useful since PrivEx collects data from an anonymity network where the stakes for information leakage are potentially higher and hence require greater robustness to bad actors. Finally, we note that the data collectors in SEPIA are provisioned for processing large quantities of traffic and data as they are part of the ISP infrastructure, but these conditions may not apply in a volunteer-resourced network like Tor. PrivEx has low overhead for the DCs.

The secret sharing scheme is an adaption of the scheme presented by Barthe *et al.* [3] which itself is an extension of previous works by Kursawe *et al.* [20], Jawurek *et al.* [15] and Shi *et al.* [32]. The novelty of PrivEx is that it introduces addition using additive secret shares for coercion resistance and perfect forward secrecy which these previous works do not address.

Anonymity Network Data Collection. The work by McCoy *et al.* [23] provided many insights about Tor client behaviour. Unfortunately, the method of safeguarding the privacy of the collected data was considered by the community at large to be insufficient. [34] Similarly, Diaz and Sassaman [7] provided insights about mix input traffic in Mix-stlye anonymous email networks by using actual traffic obtained from a public node. Here too, the use of actual traffic data had the potential to deanonymize clients. PrivEx ameliorates this state of affairs by providing researchers the means to collect statistical data about clients of anonymous networks in a privacy-preserving and compulsion-resistant manner.

Anonymity networks have to be careful about how they collect data about their network and users since they are in a position of power and can potentially expose the entire network. The operators of Tor currently collect network-wide bandwidth data but this data is independent of client data. They also collect client-specific network usage data from their guard and bridge nodes but not the exit nodes. The reason why it is considered safer to do the former and not the latter—in the context of protecting client anonymity—is that the guards/bridges already know who the clients that connect through them are so an adversary who compromises those nodes would not learn any extra information.

A key difference between PrivEx and the present Tor data collection environment is that in that latter, the true client statistics (aggregated at a per-country level, for example) are stored in a centralized database. PrivEx does not allow any entity to learn any real client data expect the nodes that originally collected the data.

9. FUTURE WORK

In the near future we aim to integrate PrivEx with the Tor codebase. Once this is done we hope to achieve acceptance from the Tor community and deploy PrivEx on the live Tor network and begin collecting statistics about website visits at exit nodes. We note that PrivEx is incrementally deployable: even if only a fraction of Tor exit nodes become DCs for PrivEx, we can still collect data about Tor traffic exiting through those particular nodes. Then, since we know the probabilities of each exit node being chosen, we can extrapolate to statistics for the whole network, albeit with somewhat larger error. Only exit nodes would have to change to support PrivEx, except for the optional enhancement of §4.2.1, which also requires the cooperation of entry nodes.

As a potential additional application of PrivEx, we note that while the Tor network does not typically try to hide the fact that a client is using Tor, there may be risks to revealing statistics gathered through widespread ingress data collection similar to those addressed by PrivEx of egress data collection. To address these potential risks, PrivEx can be applied to the present guard/bridge data collection process, and provide the same benefits as those that have been shown here for exit nodes.

An open question is whether PrivEx-like systems can be extended to collect data across subsets of the network. The risks are that this will give the adversary the ability to partition the data and perhaps learn something from the statistics that he should not have. If this can be done safely, one direct benefit is that we could, in a privacy-preserving manner, troubleshoot specific issues that are localized.

A limitation of PrivEx, since it is not needed for the scenarios we study, is that only a single query can be made of the database. We would like to investigate how to support multiple related queries—*e.g.*, network load or circuit latency—while maintaining PrivEx's privacy and utility features.

10. CONCLUSION

We have presented PrivEx, a decentralized system for privately collecting client statistics in anonymity networks. We have detailed two variants of PrivEx, one based on secret sharing and the other on distributed decryption. Both schemes are efficient and resilient to coercion attacks and malicious actors. We introduce noise, as defined in the differential privacy setting, into our aggregation process to prevent information leakage that would otherwise occur when the statistics are published.

We have used Tor as a case study and show how it can incorporate PrivEx; other anonymity networks can similarly deploy PrivEx. In this case study we collect statistics about client destination visits at the DC nodes. We show that this can be done in an efficient manner with low computational and communication overhead for conditions typical in the Tor network.

With PrivEx, our aim is to convince administrators and users of anonymity networks that client data collection *is* possible while maintaining anonymity and privacy. The benefits are that valuable information about usage trends will help guide performance and maintenance efforts. From the research perspective the benefits will be more accurate usage statistics, client models, and clearer indicators of future directions that anonymous communications and censorship resistance research should take.

Acknowledgements We would like to thank NSERC, ORF, and The Tor Project for funding this research and our CrySP lab colleagues for their invaluable feedback.

11. REFERENCES

[1] M. Alsabah, K. Bauer, T. Elahi, and I. Goldberg. The Path Less Travelled: Overcoming Tor's Bottlenecks with Traffic Splitting. In *Proceedings of the 13th Privacy Enhancing Technologies Symposium (PETS)*, pages 143–163. Springer, July 2013.

[2] Anonymizer Inc. Anonymizer. `https://www.anonymizer.com/index.html`, 2013. Retrieved May 2014.

[3] G. Barthe, G. Danezis, B. Grégoire, C. Kunz, and S. Zanella-Béguelin. Verified computational differential privacy with applications to smart metering. In *26th IEEE Computer Security Foundations Symposium (CSF)*, pages 287–301, 2013.

[4] A. Beimel, K. Nissim, and E. Omri. Distributed private data analysis: Simultaneously solving how and what. In *Advances in Cryptology–CRYPTO 2008*, pages 451–468. Springer, 2008.

[5] J. Benaloh. Dense probabilistic encryption. In *Proceedings of the Workshop on Selected Areas in Cryptography*, pages 120–128, 1994.

[6] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *19th USENIX Security Symposium*, August 2010.

[7] C. Diaz, L. Sassaman, and E. Dewitte. Comparison between two practical mix designs. In *ESORICS 2004*, pages 141–159. Springer, 2004.

[8] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[9] C. Dwork. Differential privacy. In *Automata, languages and programming*, pages 1–12. Springer, 2006.

[10] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006*, pages 486–503. Springer, 2006.

[11] C. Dwork, G. N. Rothblum, and S. Vadhan. Boosting and differential privacy. In *51st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 51–60. IEEE, 2010.

[12] T. Elahi, G. Danezis, and I. Goldberg. PrivEx: Private Collection of Traffic Statistics for Anonymous Communication Networks. In *21st ACM Conference on Computer and Communications Security*, November 2014.

[13] V. Goyal, I. Mironov, O. Pandey, and A. Sahai. Accuracy-privacy tradeoffs for two-party differentially private protocols. In *Advances in Cryptology–CRYPTO 2013*, pages 298–315. Springer, 2013.

[14] M. Hardt and A. Roth. Beating randomized response on incoherent matrices. In *44th Symposium on Theory of Computing (STOC)*, pages 1255–1268. ACM, 2012.

[15] M. Jawurek and F. Kerschbaum. Fault-tolerant privacy-preserving statistics. In *12th Privacy Enhancing Technologies Symposium (PETS)*, pages 221–238. Springer, 2012.

[16] JonDo Inc. JonDonym. `http://anonymous-proxy-servers.net/`, 2013. Retrieved May 2014.

[17] jrandom (Pseudonym). Invisible internet project (i2p) project overview. `https://geti2p.net/_static/pdf/i2p_philosophy.pdf`, August 2003. Retrieved May 2014.

[18] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography*, pages 457–476. Springer, 2013.

[19] S. Köpsell and U. Hillig. How to Achieve Blocking Resistance for Existing Systems Enabling Anonymous Web Surfing. In *Workshop on Privacy in the Electronic Society (WPES)*, Washington, DC, USA, October 2004.

[20] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *11th Privacy Enhancing Technologies Symposium (PETS)*, pages 175–191. Springer, 2011.

[21] H. Lipmaa. On the CCA1-Security of Elgamal and Damgård's Elgamal. In *Inscrypt 2010*, pages 18–35. Springer, 2010.

[22] K. Loesing. Measuring the Tor Network. `https://research.torproject.org/techreports/directory-requests-2009-06-25.pdf`, 2009. Retrieved August 2014.

[23] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker. Shining light in dark places: Understanding the Tor network. In *8th Privacy Enhancing Technologies Symposium (PETS)*, pages 63–76. Springer, 2008.

[24] A. McGregor, I. Mironov, T. Pitassi, O. Reingold, K. Talwar, and S. Vadhan. The limits of two-party differential privacy. In *51st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 81–90. IEEE, 2010.

[25] I. Mironov. On significance of the least significant bits for differential privacy. In *2012 ACM Conference on Computer and Communications Security (CCS)*, pages 650–661. ACM, 2012.

[26] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan. Computational differential privacy. In *Advances in Cryptology-CRYPTO 2009*, pages 126–142. Springer, 2009.

[27] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *2005 IEEE Symposium on Security and Privacy*. IEEE, May 2005.

[28] L. Øverlier and P. Syverson. Locating hidden servers. In *2006 IEEE Symposium on Security and Privacy*. IEEE, May 2006.

[29] J. M. Pollard. Monte carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978.

[30] K. Poulsen. Edward Snowden's Email Provider Shuts Down Amid Secret Court Battle. `http://www.wired.com/2013/08/lavabit-snowden/`, 2013. Retrieved May 2014.

[31] D. Shanks. Class number, a theory of factorization, and genera. In *Proc. Symp. Pure Math*, volume 20, pages 415–440, 1971.

[32] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *Network and Distributed System Security Symposium (NDSS)*, 2011.

[33] R. Singel. Encrypted E-Mail Company Hushmail Spills to Feds. `http://www.wired.com/threatlevel/2007/11/encrypted-e-mai/`, 2007. Retrieved May 2014.

[34] C. Soghoian. Enforced Community Standards for Research on Users of the Tor Anonymity Network. In *2nd Workshop on Ethics in Computer Security Research (WECSR)*, pages 146–153, 2011.

[35] The Tor Project. Tor Mertics Portal: Network, Advertised bandwidth distribution. `https://metrics.torproject.org/network.html`, 2014. Retrieved May 2014.

[36] The Tor Project. Tor Mertics Portal: Users. `https://metrics.torproject.org/users.html`, 2014. Retrieved May 2014.

[37] P. Winter. Towards a Tor Censorship Analysis Tool. `https://blog.torproject.org/category/tags/measurement`, 2013. Retrieved August 2014.

APPENDIX

A. SECURITY PROOF FOR PrivEx-D2

In this appendix, we show that the PrivEx-D2 scheme from Section 4.2 (using group $\mathbb{G}$ of order q with generators g and h) is secure if ElGamal encryption (using the same group $\mathbb{G}$ with generator g) is IND-CCA1. The latter fact is known to be true under reasonable assumptions [21], which establishes the security of PrivEx-D2.

The security property we seek is this: even if some of the DCs and all but one of the TKSs are adversarial, the adversary will (for each website under consideration) learn no information about the counts of the individual honest DCs, save for their sum.

We do this with a typical real-or-random game. Because the protocol uses non-interactive zero-knowledge proofs based on the Fiat-Shamir heuristic, the proof is in the random oracle model.

We denote the number of DCs by N, of which n are honest, and the number of TKSs by M, of which only number 1 is honest. The adversary game $\mathcal{G}_0$ against the PrivEx-D2 protocol proceeds as follows:

Setup phase. S1: The adversary receives the honest TKS's public key A_1 from the challenger, along with a non-interactive zero-knowledge proof of knowledge (NIZKPK) of the corresponding private key a_i such that $A_1 = g^{a_1}$. S2: The adversary then outputs the adversarial TKSs' public keys $A_2, \ldots, A_M$, along with the corresponding NIZKPKs of a_j such that $A_j = g^{a_j}$ for $j = 2, \ldots, M$.

Counting phase. C1: The adversary supplies one plaintext p_i for each honest DC ($i = 1, \ldots, n$). C2: The challenger chooses a uniformly random bit b. If $b = 0$, the challenger sets $p'_i = p_i$ for each i. If $b = 1$, the challenger picks uniformly random $p'_i \in_R \mathbb{Z}_q$ under the single constraint that $\sum_i p'_i = \sum_i p_i$.

Aggregation phase. A1: The challenger computes its ciphertexts $\langle (g^{r_i}, A^{r_i} \cdot h^{p'_i}) \rangle_{i=1}^{n}$, where $A = \prod_j A_j$ and each r_i is uniform random from $\mathbb{Z}_q$. The challenger sends commitments to these ciphertexts to the adversary. A2: The adversary selects the ciphertexts for the adversarial DCs arbitrarily, and multiplies them to yield the single ciphertext (x, y). It outputs a commitment to (x, y) to the challenger. A3: The challenger opens its commitments by sending $\langle (g^{r_i}, A^{r_i} \cdot h^{p'_i}) \rangle_{i=1}^{n}$ to the adversary. A4: The adversary opens its commitment by sending (x, y) to the challenger. A5: The challenger computes the product α of the first components of all the openings as $\alpha = x \cdot \prod_i g^{r_i}$, and returns α^{a_1} to the adversary, along with the NIZKPK of equality of discrete logarithms that $\log_g A_1 = \log_\alpha \alpha^{a_1}$. Note that $\alpha^{a_1} = x^{a_1} \cdot A_1^{\sum_i r_i}$.

Output phase. The adversary now ouputs its guess b' for the value of b. That is, it tries to decide whether the ciphertexts output in step A3 corresponded to the plaintexts supplied in step C1, or to random plaintexts with the same sum. The advantage of the adversary is $\left|Pr[b' = b] - \frac{1}{2}\right|$.

We now construct game $\mathcal{G}_1$ such that the advantage of the adversary in winning game $\mathcal{G}_1$ is the same as that of it winning game $\mathcal{G}_0$ in the random oracle model. To do this, we observe that the challenger can program the random oracle to forge any NIZKPK it creates, and can use the NIZKPK extractor to learn the adversary's private values for any NIZKPKs created by the adversary. Therefore, in game $\mathcal{G}_1$, we simply remove the challenger's NIZKPKs from steps S1 and A5, and change step S2 so that the adversary outputs the *private* keys $a_2, \ldots, a_M$. In addition, the binding and hiding properties of the commitment mean that the adversary has to compute its (x, y) before seeing the challenger's ciphertexts. Therefore, we can rearrange the steps of the Aggregation phase so that we remove the commitment steps A1 and A2, and swap the order of A3 and A4.

Now suppose an adversary $\mathcal{A}$ has non-negligible advantage in game $\mathcal{G}_1$. We next, using $\mathcal{A}$ as a black box, construct an adversary $\mathcal{B}$ for the IND-CCA1 game for ElGamal that has the same advantage. The IND-CCA1 game for ElGamal is as follows.

E1: The challenger $\mathcal{E}$ chooses a private key e uniformly at random from $\mathbb{Z}_q$, and outputs the public key $E = g^e$. E2: The adversary $\mathcal{B}$ constructs some (polynomial) number of ciphertexts (α_i, β_i) and sends them to $\mathcal{E}$. E3: $\mathcal{E}$ decrypts the ciphertexts and returns the plaintexts β_i/α_i^e to $\mathcal{B}$. E4: $\mathcal{B}$ chooses two plaintexts $m_0, m_1 \in \mathbb{G}$ and sends them to $\mathcal{E}$. E5: $\mathcal{E}$ chooses a bit b_e uniformly at random, and sends an encryption $(g^r, E^r \cdot m_{b_e})$ of m_{b_e} to $\mathcal{B}$, where $r \in_R \mathbb{Z}_q$. E6: $\mathcal{B}$ outputs its guess b'_e for the value of b_e. The advantage of $\mathcal{B}$ is $\left|Pr[b'_e = b_e] - \frac{1}{2}\right|$.

Here is how $\mathcal{B}$, acting as the challenger to adversary $\mathcal{A}$ for game $\mathcal{G}_1$, can win the above IND-CCA1 game. In step E1, $\mathcal{E}$ sends its public key E to $\mathcal{B}$. $\mathcal{B}$ sends $A_1 = E$ to $\mathcal{A}$ in step S1. In step S2, $\mathcal{A}$ outputs $a_2, \ldots, a_M$ to $\mathcal{B}$. Let $\hat{a} = \sum_{j=2}^{M} a_j$, and let $A = \prod_{j=1}^{M} A_j = E \cdot g^{\hat{a}}$.

Now in step C1, $\mathcal{A}$ supplies $p_1, \ldots, p_n$ and in step A3, $\mathcal{A}$ supplies (x, y), both to $\mathcal{B}$. Now $\mathcal{B}$ turns back to $\mathcal{E}$ and submits $(x^{-1}, 1)$ as a ciphertext in step E2; $\mathcal{E}$ will compute $1/x^{-e} = x^e$ and return it to $\mathcal{B}$ in step E3. $\mathcal{B}$ now sets $m_0 = 1$, picks $\lambda \in_R \mathbb{Z}_q^*$, sets $m_1 = h^\lambda$, and submits (m_0, m_1) to $\mathcal{E}$ in step E4. In step E5, $\mathcal{E}$ returns $(R, S) = (g^r, E^r \cdot m_{b_e})$ to $\mathcal{B}$.

$\mathcal{B}$ now needs to compute its ciphertexts $\langle (g^{r_i}, A^{r_i} \cdot h^{p'_i}) \rangle_{i=1}^{n}$ to send to $\mathcal{A}$ in step A4, such that the p'_i equal the p_i if $b_e = 0$, and the p'_i are random, but with the same sum as the p_i, if $b_e = 1$. (That is, $\mathcal{B}$ implicitly sets $b = b_e$.) To do this, $\mathcal{B}$ picks $s_i, \ldots, s_n$

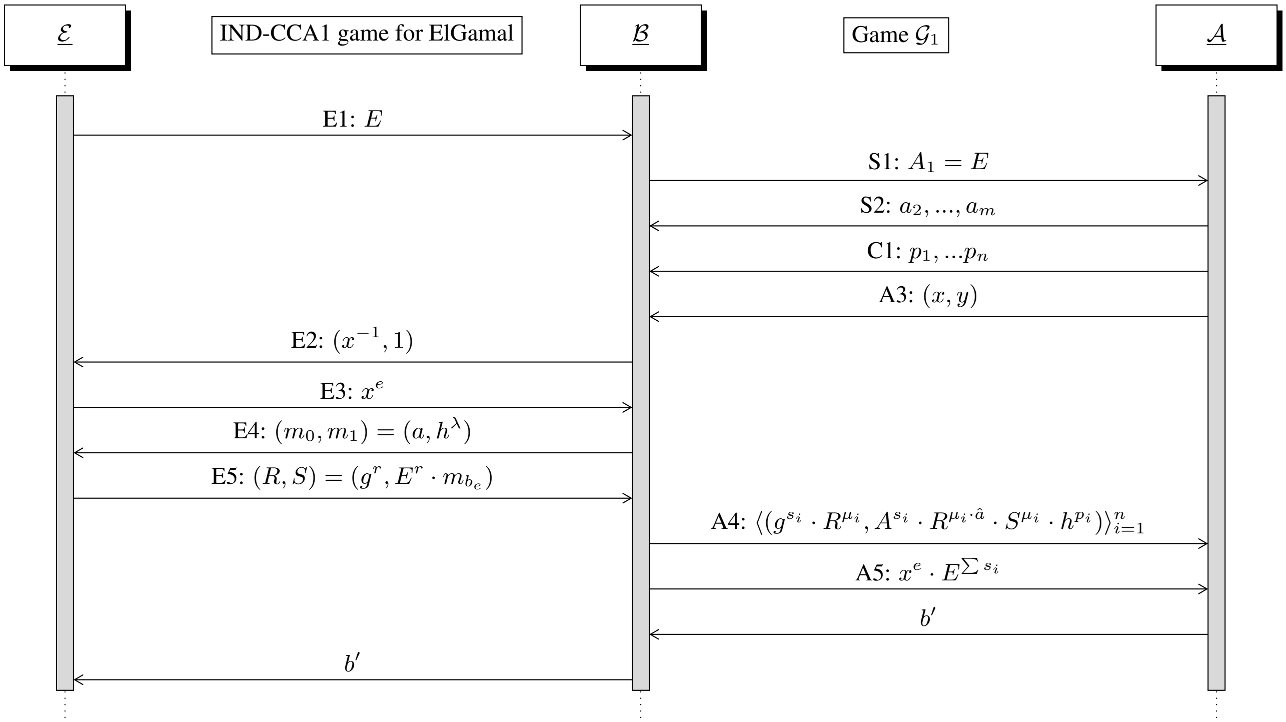

Figure 7: $\mathcal{B}$ using adversary A for game $\mathcal{G}_1$ to win the IND-CCA1 game for ElGamal against $\mathcal{E}$.

uniformly at random from $\mathbb{Z}_q$, and picks $\mu_1, \ldots, \mu_n$ uniformly at random from $\mathbb{Z}_q$ subject to the condition that $\sum \mu_i = 0$.

Now let $(R_i, S_i) = (g^{s_i} \cdot R^{\mu_i}, A^{s_i} \cdot R^{\mu_i \cdot \hat{a}} \cdot S^{\mu_i} \cdot h^{p_i})$. If $b_e = 0$, so that $m_{b_e} = 1$, we have that $(R_i, S_i) = (g^{s_i + r\mu_i}, A^{s_i} \cdot g^{r\mu_i \hat{a}} \cdot E^{r\mu_i} \cdot h^{p_i}) = (g^{s_i + r\mu_i}, A^{s_i + r\mu_i} \cdot h^{p_i})$, as required. On the other hand, if $b_e = 1$, so that $m_{b_e} = h^\lambda$, we have that $(R_i, S_i) = (g^{s_i + r\mu_i}, A^{s_i} \cdot g^{r\mu_i \hat{a}} \cdot E^{r\mu_i} \cdot h^{p_i + \lambda\mu_i}) = (g^{s_i + r\mu_i}, A^{s_i + r\mu_i} \cdot h^{p_i + \lambda\mu_i})$. Since $\sum_i \mu_i = 0$, the $p_i + \lambda\mu_i$ are random values in $\mathbb{Z}_q$ summing to $\sum_i p_i$, again as required. $\mathcal{B}$ then sends $\langle (R_i, S_i) \rangle_{i=1}^n$ to $\mathcal{A}$ in step A4.

$\mathcal{B}$'s last move is then to send $(x \cdot \prod R_i)^e$ to $\mathcal{A}$ in step A5. It can easily compute this, as it retrieved x^e from $\mathcal{E}$ in step E3, and $(\prod R_i)^e = E^{\sum s_i + r\mu_i} = E^{\sum s_i}$ since $\sum \mu_i = 0$.

Finally, $\mathcal{A}$ guesses the value of b, and since $b = b_e$, $\mathcal{B}$ simply passes that guess on to $\mathcal{E}$ as its own guess for b_e, winning the IND-CCA1 game for ElGamal if and only if $\mathcal{A}$ wins game $\mathcal{G}_1$.