Safely Measuring Tor

Rob Jansen and Aaron Johnson
U.S. Naval Research Laboratory
Washington, D.C.
{rob.g.jansen, aaron.m.johnson}@nrl.navy.mil

ABSTRACT

Tor is a popular network for anonymous communication. The usage and operation of Tor is not well-understood, however, because its privacy goals make common measurement approaches ineffective or risky. We present PrivCount, a system for measuring the Tor network designed with user privacy as a primary goal. PrivCount securely aggregates measurements across Tor relays and over time to produce differentially private outputs. PrivCount improves on prior approaches by enabling flexible exploration of many diverse kinds of Tor measurements while maintaining accuracy and privacy for each. We use PrivCount to perform a measurement study of Tor of sufficient breadth and depth to inform accurate models of Tor users and traffic. Our results indicate that Tor has 710,000 users connected but only 550,000 active at a given time, that Web traffic now constitutes 91% of data bytes on Tor, and that the strictness of relays' connection policies significantly affects the type of application data they forward.

1. INTRODUCTION

The Tor network [10] is among the most popular tools for digital privacy. Users such as journalists, activists, and law enforcement use its onion-routing protocol to keep private what Web sites they are visiting, media they are watching, and emails they are sending. As of May 2016, the network consists of around 7000 volunteer-run *relays* that forward user traffic to its destination, the network collectively forwards nearly 75 Gbps of traffic, and an estimated 1.75 million users connect every day [4].

Statistics such as these are critical to understand the impact Tor is currently having and how it can be improved. However, typical methods for network monitoring cannot be directly applied to Tor because of its strong privacy goals. For example, simply measuring the number of users is very difficult as Tor is designed to keep users anonymous, and information about their identities should not be collected even when available to protect against legal or technical compromise of the data. Tor itself currently gathers few measurements and many of these using heuristic techniques of unknown accuracy (including its number of users).

The research community has largely left this problem alone due to the privacy risks involved. McCoy *et al.* [23] in 2008 performed some of the first research measuring Tor by running a relay and examining the types of traffic observed exiting Tor. However, this approach was widely debated [28] due to the risks that collecting such data poses to user privacy. Some studies in 2010 [8, 22] performed measurement of Tor's users and traffic, but there has otherwise been relatively little follow-up. Over the same time great advances in privacy-preserving data publishing have developed, including differential privacy [11] and practical privacy-preserving aggregation [6].

We build on this recent work to develop PrivCount, an efficient and flexible system for privacy-preserving measurement on Tor. PrivCount extends the PrivEx system of Elahi *et al.* [14], designed specifically for private Tor measurement, by making it suitable for the kinds of exploratory measurements used during research. To PrivEx, PrivCount adds repeatable measurement phases in order to make iterative measurements feasible. It also provides a comprehensive method to apply differential privacy to multiple and diverse Tor statistics while maintaining high accuracy for each. We develop an open-source tool implementing PrivCount that is robust, secure, and particularly convenient to use for research on Tor [1].

We use the PrivCount tool to perform a measurement study of Tor users and traffic with a scope similar to past work but with user privacy as a foremost consideration. Our research deployment of PrivCount involves 6 independent contributors in 4 different countries of which all would need to be compromised in order to violate the security properties of the system. We perform aggregation of measurements across 7 Tor relays, which prevents any one from being identified as the source of a particular traffic characteristic.

We collect client and destination statistics with the particular goal of informing future models of Tor traffic and network improvement. Our results on exit traffic indicate that traffic to Web ports now constitutes 91% of data bytes on Tor, up from 42% in 2010 [8]. We also provide a first estimate of the number of Tor users outside of Tor's own measurement using an entirely different method. Our data indicate that in any given 10 minutes an average of 710 thousand users are connected to Tor, of which just over 550 thousand (77%) are active. We also look at the effect of *exit policies*, which relays use to limit which ports and IPs they will connect to, and

CCS'16, October 24 - 28, 2016, Vienna, Austria

ACM ISBN 978-1-4503-4139-4/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978310

we provide evidence that exit policies significantly affect the type of traffic exiting a relay, a factor that previous studies did not consider.

Our results show that many of the Tor measurements of most interest can be done privately and practically. Low-cost research efforts can productively use PrivCount, while the Tor Project itself could benefit from its security and privacy properties that exceed in many ways the security of Tor's own measurement methods.

2. PRIVATE COUNTING WITH PrivCount

In order to safely gather statistics from the Tor network, we designed and implemented a privacy-preserving data collection and aggregation system called PrivCount that expands upon the secret-sharing variant of PrivEx [14] (*i.e.* PrivEx-S2). This section provides a self-contained description of PrivCount, noting throughout how it differs from PrivEx (see Section 6 for a summary of these differences).

2.1 Overview of Roles and Architecture

PrivCount is a distributed counting system which relies on multiple nodes and entities to achieve privacy and security. A PrivCount deployment contains a *tally server* (TS) node, one or more *data collector* (DC) nodes, and one or more *share keeper* (SK) nodes.

Tally Server. The PrivCount TS is the central point of the system. The TS authenticates the DC and SK nodes before admitting them, tracks their availability and status, and synchronizes system operations. In order to minimize the attack surface of a PrivCount deployment, the TS acts as a central proxy for all communication between the other nodes. The TS server port is the only port that is required to be open and Internet-accessible in the entire PrivCount system; the DCs and SKs only make outgoing connections to the TS. This centralized control is in contrast to PrivEx, where SKs and DCs run autonomously and communicate directly. However, despite its more central position, the TS is still *untrusted*, and all communication between DCs and SKs is encrypted and authenticated.

Data Collectors. PrivCount DCs are the main nodes for processing measurements. DCs are responsible for collecting events from a locally running Tor process, computing statistics from those events, and maintaining counts for each statistic over time. Each counter is initialized with the sum of normally-distributed random noise and uniformly-random numbers shared pairwise with each SK in the system. The noise serves to provide differential privacy [11] of the final, aggregated value. The shared numbers serve to "blind" the counter; the numbers are encrypted, one for each SK, and then each is sent to the corresponding SKs (through the TS proxy) before being securely erased at the DC. This process ensures *forward privacy*: any compromise of a DC will not leak past local counter values (the counters will appear uniformly random). The DCs increment the counters during the collection period, and then send the final counts (the sum of the true counts, noise, and shared numbers) to the TS during aggregation.

Share Keepers. PrivCount SKs are responsible for storing the shared numbers assigned to it by the DCs; for each counter, each SK will have received one shared value from each DC. During aggregation, each SK for each counter sums the shared values they received from the DCs and sends the sum to the TS. Once the TS receives all counts from the DCs and all summed shared values from the SKs, the TS sums the counts from all DCs for each counter and then "de-blinds" each counter by subtracting the summed shared values. The final aggregated count for each counter is only meaningful after all summed secrets from all SKs are removed. As long as at least one SK acts honestly in summing the secret numbers, the TS cannot learn individual DC counts, and nothing is revealed but the final aggregated count, which is protected under differential privacy.

2.2 Protocol Specification

Consider a PrivCount deployment with one tally server $\mathcal{T}$, n data collectors $\mathcal{D}_i$, $i \in \{1, ..., n\}$, and m share keepers $\mathcal{S}_j$, $j \in \{1, ..., m\}$. Further, suppose that l statistics are being collected; let s_k be the value computed for the kth statistic, $k \in \{1, ..., l\}$. As in PrivEx, the value of a statistic can be a single integer, but PrivCount also allows the value of a statistic to be a vector of integers representing a histogram. Single numbers are implemented with one *counter*, which is an integer and is the fundamental data structure in PrivCount. Histograms are implemented with c_k counters $s_k = (s_{k,1}, \ldots, s_{k,c_k})$, where each counter represents a *bin* in the histogram. The range of each bin $s_{k,b}$ is $[L_{k,b}, R_{k,b})$, where $L_{k,b} < R_{k,b} \leq L_{k,b+1}$ for all b. When a measurement is made for s_k with value x, the bin $s_{k,b}$ for which $L_{k,b} \leq x < R_{k,b}$ is incremented by one. We extend some of this notation to those s_k that are single numbers by letting $c_k = 1$ and letting $s_k = s_{k,1}$. The counters are kept at all DCs, and we denote by $s^i_{k,b}$ the counter stored at $\mathcal{D}_i$ for $s_{k,b}$. Addition at a counter is performed modulo q, where q is a number large enough that the aggregated noisy value is almost certainly within $(-q/2, q/2)$. Output values in $[q/2, q)$ are interpreted as negative values, which can occur even for non-negative statistics due to the random noise added.

The goal of PrivCount is to privately measure all statistics locally and then reveal only the differentially-private counts aggregated across all DCs, *i.e.*, to compute $\sum_{i=1}^{n} s^i_{k,b} + N^i_{k,b}$, where $N^i_{k,b}$ is $\mathcal{D}_i$'s component of the random noise that provides (ϵ, δ)-differential privacy [12]. PrivCount divides the steps accomplishing this aggregation into phases that permit an iterative collection process in which initial results can be used to inform later rounds of data collection. In contrast to PrivEx, which assumes continuous and static data collection, this design enables flexibility in measurement type and length with minimal coordination among the independent parties running the protocol, while still maintaining its security properties. Because human operators facilitating the data exploration are a key component of this process, we include in our description how these operators interact with the automated system. The phases of operation for PrivCount are as follows:

2.2.1 *Initialization*

The initialization phase contains all actions that require the participation of the DC and SK operators. Coordination among multiple operators is difficult and time-consuming, and so PrivCount concentrates their involvement at the beginning phase, which allows later phases to be performed multiple times at the discretion of just the TS operator. Thus the TS operator is able to quickly and conveniently explore interesting phenomena in the Tor network without becoming a single point of compromise for the system.

We assume that a public-key infrastructure (PKI) securely provides signing and encryption keys to all PrivCount participants. These keys are used implicitly throughout the protocol when a message is encrypted to or signed by a party, and all messages are sent on secure channels implemented via these keys. Moreover, we assume that code correctly implementing PrivCount has already been installed on each protocol entity. Note that at several points individual operators are required to verify that certain settings provide "adequate" privacy and security. This determination would be made in accordance with an out-of-band discussion among all participants in the specific PrivCount deployment.

During this phase, the TS operator should configure the TS with a *deployment document* that will be shared among all protocol participants. The deployment document includes the public keys of the desired DCs and SKs. It also includes the following parameters governing the amount of privacy-preserving noise to add to the statistics: the differential privacy parameters ϵ and δ, the *sensitivity* of each statistic (*i.e.* the maximum amount the statistic can change given a certain limited amount of change in a user's activities), the *reconfiguration time* that must pass between collection periods with different configurations, and the *noise weight* w_i (*i.e.* the relative amount of noise provided by DC $\mathcal{D}_i$). These parameters are described fully in Section 2.3. The deployment document also includes the minimal subsets of DCs that must provide measurements for the SKs to allow an aggregate output to be produced. The TS then sends the deployment document to each DC and SK in the document.

After receiving the deployment document, to ensure consistency each party sends the deployment document (with the TS signature) to the other parties and waits until it has received an identical copy of the document from them. The operators should then verify that the participants, noise parameters, and the minimal required DC subsets provide adequate privacy. The acceptance or rejection is then sent to the TS.

The keys and parameters configured during the initialization phase will not be modified in later phases, and doing so requires re-initializing the system. We note that, to make larger deployments efficient, consistency of the deployment document could instead be achieved by obtaining signatures from a small set of fixed authorities (*e.g.* the Tor Directory Authorities).

2.2.2 Configuration

The TS should only proceed to the configuration phase if all DCs and SKs accepted the deployment document. During the configuration phase, the TS sets features of data collection to which the DCs and SKs automatically adjust in order to maintain PrivCount's privacy guarantees. Therefore, the TS operator can unilaterally and repeatedly configure new measurements without the TS becoming a single point of compromise, which significantly improves the speed and convenience of using PrivCount compared to PrivEx. The following features are set at the TS during this phase: (*i*) the start and end time of the collection, (*ii*) the statistics s_k that will be collected (*iii*) the number c_k of counters used to count each statistic, (*iv*) the range $[L_{k,b}, R_{k,b})$ of each bin, and (*v*) an *estimated value* v_k for each statistic that will guide PrivCount in determining how to add noise such that relative accuracy per-statistic is maximized while providing (ϵ, δ)-differential privacy across all statistics (see Section 2.3).

These configuration features are collected in a *configuration document*, which is sent to each DC and SK. To ensure consistency, each SK sends a copy of the signed document to each other SK and each DC. Each SK and each DC waits until it has received an identical copy of the configuration document from each SK. Once received, each DC then determines the noise magnitude σ_k for each statistic s_k. This calculation is based on the noise parameters set during initialization as well as the statistics selected and estimated by $\mathcal{T}$ during configuration (see Section 2.3 for details on computing the σ_k).

2.2.3 Execution

The execution phase is divided into setup, collection, and aggregation processes.

Setup. During the setup process, each DC $\mathcal{D}_i$ produces noise for each counter $s^i_{k,b}$ by sampling from the normal distribution with mean 0 and standard deviation $w_i\sigma_k$ to produce a *noise* value $N^i_{k,b} \sim \mathsf{Round}(\mathsf{Normal}(0, w_i\sigma_k))$.[1] Then $\mathcal{D}_i$ initializes the counter as $s^i_{k,b} = N^i_{k,b} \mod q$. The noise values, when aggregated across all DCs, implement the Gaussian mechanism [12] and thus provide differential privacy for the final output (see Section 3 for a privacy analysis of PrivCount).

Each $\mathcal{D}_i$ then, for each counter, generates m uniformly random integers to produce for each SK $\mathcal{S}_j$ a value $B^{i,j}_{k,b} \sim \mathsf{Uniform}(\{0, \dots, q-1\})$[2] that will be shared with $\mathcal{S}_j$, and $\mathcal{D}_i$ increments the counter by these values: $s^i_{k,b} \leftarrow s^i_{k,b} + \sum_{j=1}^{m} B^{i,j}_{k,b} \mod q$. Each counter now has $m+1$ values added to it: the noise and the m shared values. The shared values $B^{i,j}_{k,b}$ serve to blind counter $s^i_{k,b}$ to provide forward privacy and will be removed later during the aggregation process. Each $\mathcal{D}_i$ sends each shared value $B^{i,j}_{k,b}$ to $\mathcal{S}_j$ via $\mathcal{T}$ and then securely erases the locally-stored shared values. Each SK stores the received shared values for the duration of the collection process.

Collection. During the collection process, each DC monitors events from its local Tor instance and adjusts the counters. When an observation is made at $\mathcal{D}_i$ that affects statistic s_k, $\mathcal{D}_i$ adjusts one of its counters $s^i_{k,b}$. For single-number statistics, the observed number is added to the counter. For histograms, the bin $s^i_{k,b}$ in which the observation falls is incremented. For example, if the statistic is a histogram of the number of streams per circuit, and the DC observes the end of a circuit that carried 10 streams during its lifetime, then the bin containing 10 is incremented by one. The collection process lasts for the length of time set during the configuration phase.

Aggregation. Once the collection process ends, $\mathcal{T}$ requests the counter values from each DC $\mathcal{D}_i$. Each $\mathcal{D}_i$ then sends the value of each counter $s^i_{k,b}$ to $\mathcal{T}$. $\mathcal{T}$ considers as *successful* the set of DCs that provide values for all counters within a certain time, $\mathcal{S} \subseteq \{1, \dots, n\}$. Then $\mathcal{T}$ requests from each SK a sum of the values shared with the successful DCs for each counter. Recall that each SK $\mathcal{S}_j$ is storing $n \cdot \sum_{k=1}^{l} c_k$ shared values, one from each DC for each counter. If the successful

[1] Round is the nearest integer function, and $\mathsf{Normal}(\mu, \sigma)$ is the distribution with density function $\phi(\mu, \sigma; x) = e^{-(x-\mu)^2/(2\sigma^2)}/(\sigma\sqrt{2\pi})$.

[2] $\mathsf{Uniform}(S)$ is the uniform distribution over set S.

DCs are a superset of some minimal set of required DCs, as listed in the deployment document, each $\mathcal{S}_j$ adds the shared values of the successful DCs for each counter and sends the resulting sums $B^j_{k,b} = \sum_{i\in\mathcal{S}} B^{i,j}_{k,b}$ to $\mathcal{T}$. After receiving all shared-value sums $B^j_{k,b}$ from each $\mathcal{S}_j$, $\mathcal{T}$ computes the final tallies for each counter by adding the counters from the successful DCs and subtracting the corresponding shared-value sums from the SKs. That is, $\mathcal{T}$ computes $s_{k,b} = \sum_{i\in\mathcal{S}} s^i_{k,b} - \sum_{j=1}^{m} B^j_{k,b} \mod q$ and publishes these values as the output of the aggregation process. PrivCount may then be reconfigured before starting another execution phase, but it does not need to be re-initialized. In addition, the DCs and SKs will not accept a new configuration document unless it starts collection after the reconfiguration time in the deployment document has passed.

Note that tolerating the failure of DCs is of critical importance in a large distributed system such as Tor. PrivCount provides this enhancement to PrivEx at the minor cost of storing an extra $n-1$ values per counter at the SKs between setup and aggregation. We observe that SK failures can be tolerated at some cost simply by running several collections in parallel with different subsets of the SKs and choosing a final output from any of the successful SK subsets. SK failure is of much less concern, however, as the number of SKs is expected to be small even for very large deployments, and they are specifically chosen for their reliability and trustworthiness.

2.3 Privacy-Preserving Noise

PrivCount adds random noise to each counter in order to provide privacy to individual Tor user actions that contribute to its value. As in PrivEx, the formal notion of privacy used in PrivCount is (ϵ,δ)-differential privacy [12]. However, we modify and extend PrivEx in several ways to allow for an expanded set of statistics and to make it suitable for a smaller-scale research deployment.

Defining privacy. PrivCount provides privacy for a certain amount of user activity. The differential privacy guarantee applies to "adjacent" databases, where databases are typically considered to be adjacent if they differ by the input of a single user [24]. PrivEx defines adjacency as differing by at most 6 exit connections from different circuits per hour on average. This essentially protects a certain number of user connections rather than providing per-user privacy, as users may make an arbitrary number of circuits per hour. Indeed per-user privacy in Tor measurement is inherently difficult to provide accurately, as a single user can in theory constitute most of the activity being measured on the network.

We thus adopt a notion of privacy for a bounded amount activity within a given length of time and extend it to include other types of user activity. We note that Tor itself has taken a similar approach in its use of differential privacy to publish per-relay onion-service statistics [16]. The differential privacy guarantee under this notion is that, for two sets of actions within the activity bound that both occur within a certain length of time, an adversary is nearly as likely to see a given output whether a user performed one set or the other. PrivCount protects bounded numbers of the following types of user actions in a given time period: (*i*) connection to a guard, (*ii*) using a guard from a distinct IP address, (*iii*) circuit creation, (*iv*) stream creation, (*v*) sending or receiving a byte of data.

An input "database" in the context of measuring Tor is the activity on the Tor network. Given bound a_x for activity of type x and a time bound t, two sequences of network activity are then adjacent if they only differ in the actions of a single user in some time period of length t and in at most a_x actions for each type of activity x. The difference can be in the existence of such actions or in attributes of those actions. Note that these bounds apply simultaneously to different types of activity and thus can be used to provide privacy for the entirety of a user's impact on Tor if it falls within the activity and time bounds. For example, suppose that two sequences of network activity N_1 and N_2 differ only in the actions of one user in a day, and in N_2 that user both created an additional circuit and kept a stream open in N_2 for longer than it was in N_1. Then N_1 and N_2 would be considered adjacent if t is at least one day, $a_{\text{circ}} \geq 1$, and $a_{\text{stream}} \geq 1$, where a_{circ} and a_{stream} are the bounds for activities of type (*iii*) and (*iv*), respectively. We describe concrete activity bounds a_x and t in Section 4.2.

Determining noise per statistic. Given the above notion of adjacency, PrivCount provides (ϵ,δ)-differential privacy by adding normally-distributed noise to its counters, as in PrivEx. However, instead of first choosing σ to limit the adversary's "advantage" and then determining the resulting ϵ and δ, we use the more typical method of first setting ϵ and δ to achieve desired privacy and then computing the necessary σ. Tor itself is using this approach [16] (with $\epsilon = 0.3$ and $\delta = 0$), and this method makes it straightforward to allocate the privacy "budgets" ϵ and δ across statistics to minimize relative noise. Moreover, the guarantees of (ϵ,δ)-differential privacy have a direct Bayesian intepretation [27] that applies to an adversary with any amount of prior knowledge, unlike the notion of adversary advantage.

PrivCount sets ϵ and δ and then divides each among the l statistics (this division is detailed below, in the next paragraph). Let ϵ_k, δ_k be the allocation of these parameters to the kth statistic s_k, that is, $\sum_k \epsilon_k = \epsilon$ and $\sum_k \delta_k = \delta$ for $\epsilon_k, \delta_k \geq 0$. The noise to add to a statistic depends on its sensitivity. The sensitivity of a statistic that is a single number is the maximum amount that number can change between adjacent inputs. The sensitivity of a histogram is twice the number of histogram elements that can change their bin, be added, or be removed [13] (the factor of 2 appears because changing the bin of an input reduces one bin count and increases another). We describe in Section 4.2 how we determine sensitivities for the specific statistics we collected. Given the sensitivity for the kth statistic, PrivCount uses an iterative search to compute $\sigma^*(\epsilon_k,\delta_k)$, the smallest normal standard deviation such that the set of outputs that satisfy ϵ_k-differential privacy has probability at least $1-\delta_k$. PrivCount uses this value for the noise of the kth statistic: $\sigma_k = \sigma^*(\epsilon_k,\delta_k)$. This will guarantee (ϵ_k,δ_k)-differential privacy of the statistic and enable the guarantees to compose across statistics to provide (ϵ,δ)-differential privacy overall. Note that the formula given by Elahi *et al.* [14, Section 4.4.1] that relates σ_k to ϵ_k and δ_k sometimes yields parameters that do not provide differential privacy. See Appendix A for details about calculating σ_k and a counterexample to the formula of Elahi *et al.*

Allocating the privacy budget across statistics. Given total privacy budgets ϵ and δ, PrivCount divides them among the l statistics such that each statistic has high relative accuracy while providing (ϵ,δ)-differential privacy to the collective set. This improves upon PrivEx, which neither considers how the activity of a single user might affect multiple statistics nor how best to allocate a global privacy budget among the different statistics. As suggested by the bound on σ by Dwork *et al.* [12] (*viz.* $\sigma \leq \Delta\epsilon^{-1}\sqrt{2\ln(2/\delta)}$, where Δ is the sensitivity), δ affects σ far less than ϵ, and thus PrivCount simply divides it evenly: $\delta_k = \delta/l$ for all k. PrivCount then uses an estimated value v_k for each s_k to allocate ϵ such that the added noise values are expected to be proportional to the values of the statistics. If s_k represents a histogram, then v_k is the estimated total count across all bins, that is, v_k is an estimate for $\sum_b s_{k,b}$, which will help ensure that some bin has low relative noise while not requiring that good estimates are available for each bin. Estimating values can be a very effective method to improve the speed and accuracy of gathering many, diverse statistics, as the speed at which the statistics change over time can vary wildly (*e.g.* the frequency of Tor connections to Web ports is typically orders of magnitude larger than the frequency of connections to IRC ports). Estimates can be obtained from data that Tor currently publishes, from published measurement studies of Tor, or from some earlier rounds of PrivCount measurement. If no reasonable estimates are available, an equal allocation of ϵ (*i.e.* $\epsilon_k = \epsilon/l$) can be obtained simply by using the same, arbitrary value as an estimate for each statistic. Section 4.2 describes how we estimated values for our data collection.

Given the estimated values v_k, PrivCount allocates ϵ in order to minimize over all statistics the maximum ratio of the noise standard deviation to the estimated value. That is, the values ϵ_k are set to minimize $\rho = \max_k \sqrt{\sum_i w_i^2}\sigma_k/v_k$, where $\sigma_k = \sigma^*(\epsilon_k, \delta_k)$ (note that the total noise standard deviation for s_k after aggregation is $\sqrt{\sum_i w_i^2}\sigma_k$). PrivCount uses an iterative search on ρ, each value of which determines an allocation of ϵ, to compute the optimal allocation. See Appendix A for details on this computation.

Selecting noise weights. Noise with standard deviation σ_k would provide (ϵ, δ)-differential privacy across all statistics at *each* DC. However, the noise values get aggregated across DCs and thus potentially result in more noise than necessary. PrivCount therefore adjusts the noise across DCs by the noise weight w_i that is applied at each $\mathcal{D}_i$. The values w_i are used to set the desired balance between excessive noise in the aggregated result and maintaining privacy even if some DCs are malicious and do not add their component of the noise. PrivEx uses noise weight similarly and sets each w_i proportional to the consensus-weight fraction of the Tor relay monitored by $\mathcal{D}_i$ under the assumption that PrivEx runs on all Tor relays of which some fraction by consensus weight is honest. However, for smaller-scale measurement deployments, it is more appropriate to instead assume that some *number* of DCs are honest. For example, if we assume that d of n DCs are honest, then we can set $w_i = 1/\sqrt{d}$ at each $\mathcal{D}_i$.

3. SECURITY AND PRIVACY ANALYSIS

We abstract the information revealed by PrivCount with an ideal functionality in the UC-framework [7]. The initialization and configuration phases of PrivCount guarantee that there exists some set of initialization and configuration values such that each honest party either uses those values or does not participate in aggregation. This is implemented in these phases with a protocol for *broadcast with abort* [21], which is captured by the macro $\mathcal{M}_{\mathsf{BA}}$ in Figure 1. The Priv-

Macro $\mathcal{M}_{\mathsf{BA}}(sid, P, \mathcal{B})$

Start: Upon receipt of (sid, m) from P, store $(sid, P, \mathcal{B}, m, \emptyset)$.
Send: Upon input (sid, P', m) from P, if $(sid, P, \mathcal{B}, m, \mathcal{P})$ is currently stored for sid, $P' \neq P$, and $P' \notin \mathcal{P}$ then send m to P' and update the stored value to $(sid, P, \mathcal{B}, m, \mathcal{P} + P')$.
Abort: Upon input $(sid, P', \perp)$ from P, if $(sid, P, \mathcal{B}, m, \mathcal{P})$ is currently stored for sid, $P' \neq P$, $P' \notin \mathcal{P}$, and some $P \in \mathcal{B}$ is corrupt, then send $\perp$ to P' and update the stored value to $(sid, P, \mathcal{B}, m, \mathcal{P} + P')$.

Figure 1: Macro for broadcast with abort from P to $\mathcal{B}$

Count functionality $\mathcal{F}_{\mathsf{PC}}$ uses the $\mathcal{M}_{\mathsf{BA}}$ macro and is given in Figure 2. Several security and privacy properties are evident from $\mathcal{F}_{\mathsf{PC}}$, and, by the following theorem, they apply to PrivCount:

THEOREM 1. *PrivCount UC-realizes $\mathcal{F}_{\mathsf{PC}}$ in the hybrid PKI model against any adversary that does not corrupt all SKs.*

PROOF. See Appendix B. □

By Thm. 1 and examining $\mathcal{F}_{\mathsf{PC}}$, we can conclude that PrivCount securely aggregates the inputs and noise added by the honest DCs as long as one SK is honest; that is, the adversary only learns their sum. We can further observe that PrivCount provides forward privacy in that the adversary does not learn the DC inputs during data collection that occurred before corruption. We can also see that the adversary can arbitrarily modify the output of the TS for $s_{k,b}$ even if it is not compromised by choosing $\Delta_{k,b}$. We note that $\mathcal{F}_{\mathsf{PC}}$ and Thm. 1 demonstrate how to define and prove privacy for PrivEx-S2 as well, which was not shown by Elahi *et al.* [14].

We can show that sufficiently stringent policies on deployment documents ensure (ϵ, δ)-differential privacy, as shown by the following theorem:

THEOREM 2. *If at least one SK $\mathcal{S}_i$ is honest, and if, for each minimal subset $\mathcal{S}$ of DCs in the deployment document that $\mathcal{S}_i$ receives with honest subset $\mathcal{H} \subseteq \mathcal{S}$, $\sqrt{\sum_{\mathcal{D}_i \in \mathcal{H}} w_i^2} \geq 1$, then the output of PrivCount is (ϵ, δ)-differentially private.*

PROOF. See Appendix B. □

We do note that differential privacy only provides privacy for a limited amount of time and user activity, and an active user will eventually exceed that amount. Thus, as the number of measurement rounds increases, the chance of revealing something private about user activity increases. For example, the existence of a regular Tor user may become apparent over time even if it cannot be identified in the statistics from any single measurement round. Goulet et al. [16] choose to round differentially-private outputs in an attempt to avoid such an information leak. This technique could be applied to PrivCount's statistics as well.

4. METHODOLOGY

In this section we describe our open-source prototype implementation of PrivCount [1] and provide details about the deployment that we set up and used to measure Tor. As the privacy of Tor users is a primary concern, we practice data

Functionality $\mathcal{F}_{\mathsf{PC}}$

Let $\mathcal{A}$ be the adversary. Copies of all inputs and outputs are sent to $\mathcal{A}$ except those with the input command. When a party is corrupted by $\mathcal{A}$, and all future inputs and outputs are with $\mathcal{A}$.
Initialization: A deployment and its TS are established by receiving TS from some party P_{TS} and P_{TS} from each other party. Then the macro $\mathcal{M}_{\mathsf{BA}}(\mathsf{init}, P_{\mathsf{TS}}, \mathcal{P})$ is run, where $\mathcal{P}$ is all parties except P_{TS}, which receives the deployment document from the TS and broadcasts it to the other parties. After $\mathcal{M}_{\mathsf{BA}}$ sends document D_{init} to P, P aborts if $D_{\mathsf{init}} = \perp$; otherwise, a bit b is accepted from P, where $b = 1$ indicates acceptance of D_{init}, and forwards it to P_{TS}.
Configuration: If each honest party has accepted D_{init} and no party is currently in execution, the macro $\mathcal{M}_{\mathsf{BA}}(\mathsf{config}, P_{\mathsf{TS}}, \mathcal{P}_{\mathsf{SK}})$ is run, where $\mathcal{P}_{\mathsf{SK}}$ is the set of SKs, which receives a new configuration document from P_{TS} and broadcasts it to the other parties.
Execution setup: Once P_i has received a configuration document $D_{\mathsf{config}} \neq \perp$, for each statistic s_k, based on the values in D_{init} and D_{config}, c_k counters are created, and the noise magnitude σ_k is computed, and the counters are each initialized with random noise sampled from $\mathsf{Normal}(0, w_i\sigma_k)$ and rounded. At this point SKs in the D_{init} skip collection and enter aggregation.
Execution collection: Upon receiving (start, P_i) from P_{TS}, if P_i has completed setup and is a DC in the D_{init}, then it enters data collection. While in data collection, P_i accepts inputs $(\mathsf{input}, P_i, k, x)$ as long as P_i is not corrupt. If statistic s_k is a single number, then its counter is increased by x, and if it is a histogram, the bin b such that $L_{k,b} \leq x < R_{k,b}$ is incremented by one. If P_i becomes corrupt, then its counters cease to be incremented. Upon receiving (stop, P_i) from P_{TS}, a P_i in data collection ends execution and waits for the next configuration document.
Execution aggregation: Upon receiving $(\mathsf{share}, \mathcal{S}_i, P_i)$ from P_{TS}, if P_i is in aggregation, then P_i leaves execution and waits for the next configuration document. Then, once all honest parties have left execution, suppose that the honest DCs in $\mathcal{S}_i$ are the same for all the honest SKs and each $\mathcal{S}_i$ contains some minimal subset of required DCs in D_{init}. Let $y_{k,b}$ be the sum of the counters $s^i_{k,b}$ summed over the DCs $\mathcal{D}_i$. If P_{TS} is corrupt, then the $y_{k,b}$ are output to the adversary, and otherwise, upon receiving $\Delta_{k,b}$ from the adversary, $y_{k,b}$ is added to $\Delta_{k,b}$ and the values are output to P_{TS}. Suppose instead some $\mathcal{S}_i$ doesn't contain any minimal subset of required DCs. Then, if P_{TS} is corrupt, $\perp$ is output to the adversary, and otherwise $\perp$ is output to P_{TS}. Otherwise it must be that the $\mathcal{S}_i$ of two SKs disagree on honest DCs, and then a uniformly random integer mod q is output to the adversary if P_{TS} is corrupt and otherwise is output to P_{TS}.

Figure 2: Ideal functionality for PrivCount

minimization during the measurement process: we focus our collection of statistics on only those that aid Tor traffic modeling efforts [18]. Measurements from both entry and exit relay positions are considered as both positions provide unique information that is useful for modeling purposes.

4.1 Measuring Tor with PrivCount

We use PrivCount to safely measure Tor. While the PrivCount protocol was described in Section 2, we now explain how we collect and process the Tor events that will allow us to compute our statistics of interest.
Collecting Events. To facilitate the collection and processing of events from Tor, we extended the Tor control protocol [2] with a new asynchronous event of type `PRIVCOUNT`. Once an application authenticates with Tor and registers for our new event, our modified Tor software will then emit the following whenever any of the sub-events occur: (i) *exit stream end*: channel id, circuit id, stream id, num bytes read, num bytes written, exit port, start time, end time; (ii) *exit circuit end*: channel id, circuit id; (iii) *entry circuit end*: channel id, circuit id, num client-bound cells, num exit-bound cells, start time, end time, client IP; and (iv) *entry connection end*: channel id. Each PrivCount DC connects to a Tor relay running our modified software to collect this information, and processes it as follows.
Classifying Streams. In order to better understand likely similar behaviors of Tor users, DCs classify streams into traffic classes based on each stream's exit port. Whenever an *exit stream end* event is received, a DC will label the stream with one of the following traffic classes: HTTP/S ports 80 and 443 are labeled as *Web*; SSH and common IRC ports[3] are labeled as *Interactive*; and any remaining port is labeled as *Other*.

[3]22, 194, 994, [6660, 6670], 6679, 6697, and 7000

Classifying by exit port only provides a rough approximation of behaviors: the classification does not perfectly segment application protocols since different applications can use the same port. However, our method reduces the privacy risk over more accurate techniques like protocol sniffing via deep packet inspection, and does not require the processing and storage overhead of statistical learning.
Classifying Circuits. We classify circuits into those that appear to be used and those that do not. Whenever an *exit circuit end* event is received, a DC will label the circuit as *active* if at least one stream was completed on that circuit and *inactive* otherwise. Additionally, the circuit is labeled as carrying Web, Interactive, and Other traffic if at least one Web, Interactive, and Other stream completed on the circuit, respectively. Whenever an *entry circuit end* event is received, the circuit is labeled as *active* if at least 8 cells have been transferred on that circuit (6 cells for circuit setup, and at least 1 more in each direction for circuit usage), and *inactive* otherwise.
Rotating IP Address Maps. PrivCount uses a mapping of client IP address to count unique per-client statistics. Because of the sensitivity of storing client IP addresses in memory, we limit the amount of time over which we count per-client statistics before clearing the map to 20 minutes. This is comparable to the default circuit lifetime in Tor of 10 minutes, and we also note that the Tor Project takes a similar approach for maintaining statistics such as per-country user numbers but is currently storing client IP addresses over a much longer and less-safe period of 24 hours.

4.2 PrivCount Deployment

We deployed PrivCount on the live Tor network with 1 tally server, 6 share keepers, and 7 data collectors monitoring one Tor relay each. Our SKs were each run independently, with 6 different operators on 6 different machines in 4 different countries. Our DCs and Tor relays were run by 2 op-

Table 1: Relays & DC machines in PrivCount deployment

Nickname	Fingerprint	Host	Link Speed
NoneRunLong	`0x9068A1E53C98`	1	100 Mbit/s
PhantomTrain1	`0x7EE45524BCA7`	2	100 Mbit/s
PhantomTrain2	`0xF789CB84E7C6`	2	
PhantomTrain3	`0x412BDCB24295`	3	1 Gbit/s
PhantomTrain4	`0x3A8A5432C008`	3	
PhantomTrain5	`0x3C0AD8F7DFC1`	3	
PhantomTrain6	`0x653632EEF25D`	3	

Table 2: Exploratory measurement round periods

Name	Start Consensus (UTC)	End Consensus (UTC)
Strict	2016-05-17-14-00-00	2016-05-18-13-00-00
Default	2016-04-29-15-00-00	2016-04-30-14-00-00
FS	2016-05-15-02-00-00	2016-05-16-01-00-00
FS+	2016-05-13-23-00-00	2016-05-14-22-00-00
FS++	2016-05-16-05-00-00	2016-05-17-04-00-00
Open	2016-05-08-18-00-00	2016-05-09-17-00-00

Table 3: In-depth measurement round periods

Stat. Type	Start Consensus (UTC)	End Consensus (UTC)
Entry	2016-08-11-03-00-00	2016-08-15-02-00-00
Exit	2016-07-16-18-00-00	2016-08-06-17-00-00

erators on 3 different host machines (see Table 1 for details). We focused on providing a highly-secure SK infrastructure to demonstrate the feasibility of running PrivCount with *any* number of Tor relays, and indeed our SK setup requires 6 operator or machine compromises to subvert, which provides comparable security to Tor's current set of 9 Directory Authorities of which only 5 need to be compromised in order to control the network consensus and thus Tor itself. We used PGP keys as the PKI.

Collection Rounds and Statistics. We collected Tor measurements in multiple rounds of different lengths. The main constraint driving this approach is the need to add sufficient noise to obscure the impact of a user's activity across all statistics. Given the size of the relays in our deployment, collecting as few as a dozen statistics requires on the order of a day for the added noise to be less than 10% of the estimated value of each. Thus we ran multiple *exploratory* rounds of 1 day each and *in-depth* rounds of varying length. The two round types are interleaved, although most exploratory rounds came first so that their results could inform the in-depth rounds. Each collected statistic is either an *entry statistic*, that is, only incremented when the relay is in the entry position of a Tor circuit, or an *exit statistic*, that is, only incremented in the exit position. This ensures that each statistic is only incremented by the DC of one relay, thus limiting its sensitivity and reducing the added noise.

In the exploratory rounds, we collected 13 of the single-number type of statistics with the purpose of obtaining estimates of the average levels of various types of Tor activity. We present in Section 5.2 the results from running our PrivCount deployment 6 times, where each collection phase ran for 24 hours with a different exit policy (see Table 2 for details). Before starting each collection phase, we adjusted our exit relays' exit policies and verified that the new policy was correctly propagated to the official Tor consensus. Over all exploratory collection phases, the mean probability of selecting our relays in the entry position was 0.196% and the mean probability of selecting our relays in the exit position was 1.283%.

The statistics collected during the exploratory rounds are single numbers. The only entry statistic is the sum of unique client IPs observed in consecutive 10-minute time slices. The exit statistics count active and inactive circuits, streams, and data bytes sent or received. Each of these items is counted overall as well as per-class (*i.e.* Web, Interactive, and Other).

In the in-depth rounds, we focused on statistics describing the most significant types of Tor activity, as determined from the exploratory rounds. In some of the in-depth rounds we focused on entry statistics while in others we focused on exit statistics. When counting exit statistics, we used histograms to get distributions of activity instead of simply averages. We used only 3–4 bins per histogram in order to obtain sufficient relative accuracy of at least the most popular bins, given the added noise. Multiple in-depth rounds were run to adjust the bin ranges to obtain a more even distribution of the bin counts.

In Section 5.3, we present 4 entry statistics (4 single numbers) collected throughout an in-depth measurement phase that we ran for 4 days, and 26 exit statistics (10 single numbers and 16 histograms) collected throughout an in-depth measurement phase that we ran for 21 days (see Table 3 for details). The mean probability of choosing our relays in the entry position was 0.130% during the entry statistics collection phase, and the mean probability of choosing our relays in the exit position was 0.914% during the exit statistics collection phase.

The entry statistics collected during the in-depth rounds include the addition of the number of active and inactive clients over 10-minute time slices as well as the total number of client connections. The exit statistics collected include the addition of a histogram tabulating circuit lifetimes as well as histograms for circuits of the Web and Other classes measuring the number of streams each circuit carries. In addition, a histogram for inter-stream creation times is added for all streams, Web streams, and Other streams. Histograms are also added just for Web and Other streams that measure the number of bytes out to the destination, the number of bytes in to the client, and the ratio of bytes out to bytes in.

Because our exploratory measurements and analysis indicated that Interactive-type traffic is a minor part of Tor's traffic distribution, we chose not to collect Interactive-related statistics during our in-depth phases (to reduce the risk to privacy and increase accuracy of the other measurements). Relatedly, we chose to collect in-depth measurements using the default exit policy because we found that it is the most supported policy: 84.7% of exit relays accept ports 80 or 443 and block or partially block all ports that are also blocked by the default policy.

Noise. Our PrivCount deployment uses differential-privacy parameters of $\epsilon = 0.3$ and $\delta = 10^{-3}$. The ϵ value is the same used by Tor for hidden-service statistics [16]. δ can be viewed as an upper bound on the probability of choosing a noise value that violates ϵ-differential privacy.

We use a reconfiguration time of 24 hours. Table 4 gives the action bounds (see Section 2.3) that define the privacy afforded by our collection. Tor clients by default maintain one entry connection, and so we provide privacy for 24 hours worth of client observations at the entry as well as 12 total entry connections (*i.e.* a new one every 2 hours). Tor clients by default create 2 preemptive circuits and use each circuit for 10 minutes. Therefore, we protect 24 hours of continuous circuit use: $146 = 24 \cdot 60/10 + 2$. Note that we protect

Table 4: Action bounds in PrivCount deployment

Action	Bound
Simultaneous open entry connections	1
Time each entry connection open	24 hrs
New entry connections	12
New circuits	146
New Interactive circuits	20
New streams	30,000
New Interactive streams	20
New Other streams	144
Data sent or received	10 MiB

a smaller number of Interactive circuits (20), which typically are used longer and thus created less frequently than those of other classes. We protect 30,000 streams, which in particular covers 100 Web pages with at most 300 requests per page, an amount that includes 95% of Web pages measured by HTTP Archive [5]. For Interactive and Other streams, we protect only one stream for every circuit created within the action bound (20 and 144, respectively). We also protect 10MiB of traffic in either direction. This covers 95% of Web pages measured by HTTP Archive, and it is ten times the traffic amount protected in Tor's hidden-service statistics [16].

Setting the action bounds requires balancing between privacy and accuracy. To obtain reasonable accuracy, we must choose bounds that aren't high relative to actual activity on Tor (and in particular on relays in our deployment). Thus, we choose to provide stronger protection for actions that are quite frequent (*e.g.* creating a Web stream) than for those that are less frequent (*e.g.* creating an Interactive stream). This might appear inconsistent, but we argue that the privacy protections are sufficient in every case and are simply even better in some cases than others.

The sensitivities of the statistics are easily determined from these action bounds. For example, the sensitivity of the statistic counting active circuits is simply 146, the action bound for creating circuits. Note that the histograms are sensitive to the *number* of inputs that can change value and not to the change in their values. For example, the sensitivity of the histogram of inter-stream creation times is 60,000, because we allow a single user to change the lifetimes and existence of 30,000 streams, and the sensitivity of a histogram is twice the number of inputs that can change. Using per-statistic sensitivities to determine the noise standard deviation is in general a worst-case approximation that allows all statistics to *simultaneously* change by as much as their sensitivities, but for our deployment the statistics can in fact all simultaneously change by that much.

We set noise weights to provide differential privacy if at least one *machine* running DCs is honest. The 3 machines in our deployment run 1, 2, and 4 DCs (recall that each DC is paired with a Tor relay). For a machine running $k \in \{1,2,4\}$ DCs, the noise weight of its DCs is $1/\sqrt{k}$. Thus any one honest machine contributes all the necessary noise, and if all are honest then $\sqrt{3}$ times as much noise is added as is necessary. We include the entire set of DCs as the minimal set, as in our relatively small deployment losing the input of any relay would significantly affect the accuracy of our statistics.

Estimating values. For exploratory rounds, we estimated the values of the statistics using the extra-info documents published by Tor relays ([3], Section 2.1.2). These documents include per-relay statistics, including in particular data about (*i*) users seen at entries; (*ii*) circuits, including their traffic amounts; and (*iii*) streams at exits, including per-port numbers and traffic amounts. The data from these three categories are incomplete because they are all turned off by default and are thus reported by a minority of relays. Moreover, they are limited to only those statistics Tor has built in, and their accuracy is generally poor, as each relay obfuscates its data (*e.g.* rounding up to a given multiple), thus producing noise that scales with the number of relays. We also note that these statistics generally do not satisfy any formal privacy notion, and their accuracy and frequency is chosen via ad hoc privacy arguments. However, they provide enough data for us to produce reasonable estimates for each of the statistics in our exploratory rounds. We produce estimates from the extra-info values by adjusting them for our relays' weights and the length of the exploratory round, making educated guesses to fill in a few gaps (*e.g.* guessing the number of Web circuits based on the number of observed Web streams).

For the in-depth rounds, we use the results from our exploratory rounds, adjusted for the change in round length. Most of the statistics added to the in-depth over exploratory rounds are histograms. For these, we only need to estimate the total number of inputs (*i.e.* not their values), and the exploratory rounds provide such estimates.

Research Ethics. The Tor Project issues safety guidelines[4] for researchers studying the live Tor network to help them consider how their research might impact Tor users. We carefully consider these guidelines when designing our measurements. We would highlight that we practice data minimization by limiting the statistics we gather to just those needed to understand the major features of client and exit traffic for purposes of modeling Tor (*e.g.* in Shadow [18]) and improving Tor's performance. We also note that all of our measurements are designed to preserve user privacy and are safe to release publicly. Overall, measuring Tor while protecting users is a primary challenge addressed by this work.

5. MEASUREMENT RESULTS

In this section, we provide details about our measurement strategy while describing the measurement results.

5.1 Exit Policy Analysis

Each Tor relay is configured with an *exit policy* which specifies the destination ports[5] to which the relay is and is not allowed to connect. The exit policies of all relays are included in Tor consensus documents and are used by Tor clients to choose suitable exit nodes for their intended destination port. As a result, an exit relay with a very restrictive exit policy (*e.g.* one that allows only port 22) will potentially observe significantly different traffic characteristics than an exit relay that allows all ports (including ports 80 and 443).

In order to ensure that the measurements taken at our exit relays provide us with an accurate view of all Tor network traffic, we conducted an analysis of the exit policies used by Tor exit relays. After analyzing multiple consensuses that were produced during April 2016, we found that all 219 ports rejected in Tor's default exit policy[6] (most of which are re-

[4]https://research.torproject.org/safetyboard.html

[5]Our usage of exit policies only involves port numbers, although IP addresses may also be specified.

[6]reject 25, 119, [135-139], 445, 563, 1214, [4661-4666], [6346-6429], 6699, [6881-6999]; accept *

Table 5: **Ports considered for exploratory measurements**

Traffic Type	Label	Ports
HTTP/S	web	80,443
BitTorrent Base	btb	[6881-6889]
BitTorrent Extended	bte	[6890-6999]
Other File-Sharing	ofs	1214,[4661-4666],[6346-6429],6699
Various	var	25,119,[135-139],445,563

Table 6: Policies considered for exploratory measurements and weighted exit bandwidth supporting each policy

Name	Policy by Label (see Table 5)	Exit BW (%)
Strict	**reject** web, btb, bte, ofs, var	14
Default	**accept** web; **reject** btb, bte, ofs, var	68
FS	**accept** web, ofs; **reject** btb, bte, var	3.7
FS+	**accept** web, bte, ofs; **reject** btb, var	2.7
FS++	**accept** web, btb, bte, ofs; **reject** var	9.9
Open	**accept** web, btb, bte, ofs, var	1.1

lated to file-sharing) constituted the 219 most rejected ports in Tor. We also found that similar fractions of exit bandwidth are available for several of the ports that are blocked by default. Table 5 shows these similarly grouped ports and the type of traffic commonly associated with them, as well as the ports of the most accepted traffic type (HTTP/S).

We continued our analysis to understand which combinations of the groups of ports in Table 5 are accepted and rejected by each relay. We computed the percentage of exit relays, weighted by their probability of selection in the exit position, that accepted or rejected each group of ports. Of the $2^5 = 32$ possible combinations of these exit policies, only 7 of them were valid (that is, at least one exit matched the policy). Table 6 lists the top 6 policies[7] and the percentage of exit bandwidth that supported each policy (we ignore the remaining valid exit policy as it was supported by less than 0.1% of exit bandwidth).

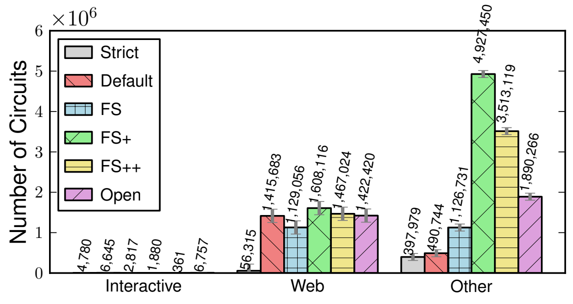
We found that the default policy is by far the most popular policy: it is supported by 68% of relays and supports exit to the 65,315 most allowed ports. We also found that 17% of exits support at least some of the ports commonly associated with file-sharing, even though those ports are among the least allowed ports. Finally, over 14% of exit bandwidth does not support exiting to the HTTP/S ports 80 and 443, although it is the most allowed traffic type in Tor. We selected the exit policies to use on our exit relays during the exploratory phase based on the results of this analysis.

5.2 Exploratory Measurements

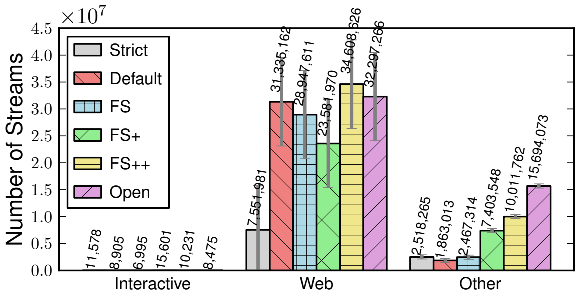
We first discuss the measurement results collected with our PrivCount deployment before describing how to use them to infer full Tor network statistics.

Results. Our primary measurement results related to traffic distribution are shown in Figure 3. Figure 3a shows the total number of active circuits (circuits with at least one completed stream) as counted by our exit relays over the 24 hour collection intervals and according to the traffic type classified by PrivCount. Figure 3b similarly shows the number of streams (TCP connections that exit the Tor network) observed by our exit relays, and Figure 3c shows the sum of the amount of data transferred in both directions on those streams. The collected measurement results for each of the exit policies from

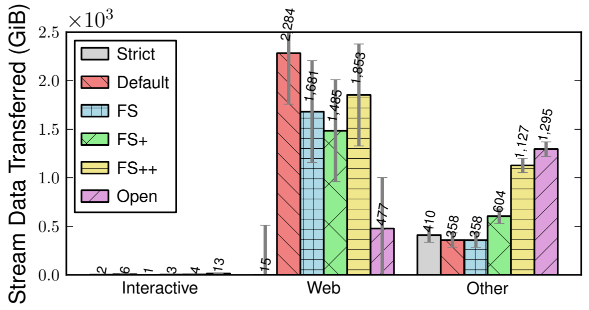
[7]All policies are prepended to Tor's default policy.



(a) Counts of active Tor circuits



(b) Counts of connections exiting Tor



(c) Counts of data transferred on connections exiting Tor

Figure 3: Measurement results collected while using various exit policies, by the traffic type classified by PrivCount. Error bars show 95 percent confidence intervals.

Table 6 are shown with 95% confidence intervals that account for the noise that was probabilistically added to each counter.

Consistent across all six exit policies and all three statistics shown in Figure 3, we observed very low counts for Interactive traffic relative to Web or Other. This indicates that Interactive traffic is a very minor part of Tor's overall traffic distribution and so may be safely ignored during later measurement periods.

We also observed that the number of circuits (3a) and the number of streams (3b) for the Web class remained relatively consistent across all of the exit policies that we used during measurement, except for the *strict* policy. The number of streams did show some fluctuation among exit policies, which we attribute to the somewhat large error bounds for the Web class. Note that the *strict* exit policy does not allow traffic that would have been classified as Web, and therefore the positive counts for Web under the *strict* policy in Figure 3 correspond directly to the noise that was added to those counters by PrivCount.

Table 7: 10-minute mean Tor network activity inferred from exploratory measurements

Traffic Class	Active Circuits ($\times 10^3$) Count	Active Circuits ($\times 10^3$) 95% CI	Streams ($\times 10^3$) Count	Streams ($\times 10^3$) 95% CI	Stream Data (GiB) Count	Stream Data (GiB) 95% CI
Total	1,200 (100%)	±500 (42%)	17,000 (100%)	±4,000 (24%)	1,100 (100%)	±300 (27%)
Interactive	3 (0%)	±2 (67%)	5 (0%)	±4 (80%)	3 (0%)	±1 (33%)
Web	640 (53%)	±80 (13%)	15,000 (88%)	±4,000 (27%)	1,000 (91%)	±300 (30%)
Other	580 (48%)	±50 (9%)	1,900 (11%)	±200 (11%)	260 (24%)	±40 (15%)

Although Web circuits and streams remain relatively consistent across exit policies, Other circuits and streams do not. The number of observed Other circuits (3a) increases dramatically and somewhat sporadically when using exit policies that allow exiting to file-sharing ports. The number of observed Other streams (3b) similarly increases when file-sharing ports are allowed, and the number of streams trends higher as less restrictive exit policies are used (and therefore more file-sharing ports are allowed). These observations follow common traffic patterns of file-sharing protocols, which tend to create many connections with their peers in order to decrease download times for large files.

Similar to the circuit and stream count trends, we observed that the amount of Other stream data transferred on connections exiting Tor (3c) also increases significantly as the number of file-sharing ports that our exit relays allow increases. However, unlike the Web circuit and stream counts (which are mostly unaffected by the increase in Other circuits and streams), the amount of Web data transferred decreases as the amount of Other data transferred increases. This suggests a crowding-out situation, where each of the traffic classes are in direct competition for the limited bandwidth of our exit relays and file-sharing traffic wins the competition. The number of Web circuits (3a) and streams (3b) are consistent even when the amount of transferred Web data (3c) decreases, indicating that the performance for the web streams that coexist with file-sharing streams may be significantly reduced. If this hypothesis holds true, then users could potentially improve their web browsing experience simply by excluding those exit relays that allow file-sharing ports in their exit policies when building circuits. This presents an interesting area for future work.

Finally, we observed that restricting Web traffic from exiting our relays using the *strict* policy did not result in a dramatic increase in circuits, streams, or data transferred for either the Interactive or Other classes. This increases our confidence that there are not other ports whose activity is crowded out by the activity associated with the Web ports.

Inferring Network Totals. Our results from the exploratory phase improve our understanding of how traffic characteristics vary with different exit policies. However, the results can be somewhat misleading when compared directly without accounting for the relative exit bandwidth support available for each exit policy type. Accounting for the popularity of each exit policy used during measurement will provide a better estimate of the overall traffic characteristics over the entire Tor network. We now produce such an inference.

We collected the 24 consensus files (one for each hour) that were produced during each of the 6 exploratory measurement rounds presented above ($24 \cdot 6 = 144$ files total). For each set of 24 files, which corresponds to a single exit policy and a collection of measurement results, we computed the mean fractional weight for selecting our relays in the entry position as W_g, the mean fractional weight for selecting our relays in the exit position as W_e, and the mean fractional weight of other relays supporting the exit policy that we used during that collection phase as W_p. We then scale our measured entry statistics by W_p/W_g and our measured exit statistics by W_p/W_e, and sum the scaled results from all phases to produce a final network total estimate.

Table 8: Comparison of exit traffic distribution

	Traffic Class	2008 [23]	2010 [8]	This Work
Conns	HTTP/S	96.51%	70.40%	88%
	Interactive	0.08%	1.72%	0%
	Other	3.41%	27.88%	11%
Bytes	HTTP/S	59.52%	41.81%	91%
	Interactive	0.23%	0.26%	0%
	Other	51.82%	57.92%	24%

Table 7 shows the inferred estimates of the number of active circuits, streams, and total amount of data transferred on streams for each traffic class and across the entire Tor network during an average 10-minute interval. Note that for the statistics that don't sum to 100%, the percentages shown in the table are calculated using a separate, single total count that we collected (*e.g.* total active circuits) rather than a sum of the three class-specific individual counts. This was done to avoid aggregating noise and improve the accuracy of the estimate where possible. The confidence intervals shown include the uncertainty associated with the noise added to the counters as well as the sampling error.

As with our direct measurements, our inference indicates that Interactive traffic is a very minor contributor to the overall inferred activity on Tor. Our inference also indicates that Web traffic is the dominant type of traffic seen on the network for all three of the statistics shown in the table. In particular, we found that Web traffic accounts for 88% of the streams created in Tor and 91% of the stream data. A comparison to previous measurement studies is given in Table 8. Overall, our results indicate that HTTP/S usage has increased since 2010 relative to other protocols, which we suggest is due to usability improvements in browsing the web with Tor (*e.g.* Tor Browser).

5.3 In-Depth Measurements

We present the results from our in-depth rounds as inferred Tor network totals that we computed by adjusting each entry and exit statistic by our mean entry and exit probability, respectively. Note that the inferences based on exit statistics assume that relays using non-default exit policies observe traffic characteristics similar to those observed while using the default policy. Future work should consider taking these measurements with different exit policies (*e.g.*, that allow or partially allow file-sharing ports).

Entry Statistics. The entry statistics that we collected focused on counting the number of observed unique clients and the number of those clients that were active and inactive. Re-

Table 9: 10-minute mean Tor network activity inferred from single-counter in-depth entry statistics

Statistic	Count	95% CI
Unique Clients ($\times 10^3$)	710 (100%)	±85 (12%)
Active	550 (77%)	±70 (13%)
Inactive	140 (20%)	±30 (21%)
Client Conns ($\times 10^3$)	560 (100%)	±60 (11%)

Table 10: 10-minute mean Tor network activity inferred from single-counter in-depth exit statistics

Statistic	Count	95% CI
Active Circuits ($\times 10^3$)	1,400 (100%)	±200 (14%)
Web	700 (50%)	±100 (14%)
Other	680 (49%)	±50 (7%)
Inactive Circuits ($\times 10^3$)	1,300 (100%)	±100 (8%)
Streams ($\times 10^3$)	15,000 (100%)	±3,000 (20%)
Web	17,000 (113%)	±3,000 (18%)
Other	1,600 (11%)	±200 (13%)
Stream Data (GiB)	1,300 (100%)	±200 (15%)
Web	900 (70%)	±200 (22%)
Other	230 (18%)	±40 (17%)

call that the unique client IP addresses were recounted over 10-minute intervals for privacy reasons. Table 9 shows the mean inferred counts over all 10-minute intervals that occurred during our 4-day entry measurement period (there are $4 \cdot 24 \cdot 6 = 576$ such intervals) as well as the confidence intervals (accounting for noise and sampling error).

We found that Tor has about 700 thousand unique clients connecting to the network during an average 10-minute interval. Compared to Tor's own estimate of about 1.75 million clients per day in May 2016 [4], this suggests that the client population turns over about 2.5 times a day. Somewhat surprisingly, we found that about 130 thousand clients have inactive circuits during an average 10 minutes. Also, we found fewer connections from clients compared to the number of unique clients, but we note that the client connections that did not close during our 4 day measurement period would not have been counted (because PrivCount counts connections when they are closed).

Exit Statistics. Our exit statistics focus on circuit and stream statistics to help better understand Tor traffic. Table 10 shows the mean inferred counts over all 10-minute intervals that occurred during our 21-day exit measurement period (there are $21 \cdot 24 \cdot 6 = 3024$ such intervals) as well as the confidence intervals (accounting for noise and sampling error). Although all of the counts were taken over the entire 21-day exit measurement period, we show the counts as 10-minute means for clarity of presentation.

We inferred that there are 1.4 million active circuits used during an average 10 minutes, which corresponds to between 2 and 3 active circuits per active client. We also found about the same number of inactive circuits as active circuits during an average 10 minutes, which may not be surprising given the large number of inactive clients we inferred from the entry statistics and also given that some of Tor's circuits that are preemptively generated for performance reasons may never be used. As in the exploratory rounds, we found that a majority of the active circuits carry Web traffic, and that an overwhelming majority of streams and stream data corresponds to Web traffic. Our inferences indicate that there are an average of about 25 Web streams per Web circuit, and that exit relays exchange an average of about 50 KiB with the destination for each such stream. Similarly, there are an average of about 2 Other streams per Other circuit, and an average of about 150 KiB is exchanged with the destination per Other stream.

The histogram statistics are shown in Table 11. For each statistic, the table shows the bin ranges and the relative counts as percentages of the number of times the statistic fell within each bin range, and the 95% confidence interval that applies to each bin count. The percentages shown were computed by taking the bin count as a fraction of the single-counter total counts (*i.e.* number of circuits, streams, and stream data) for each traffic type (*i.e.* Total, Web, and Other), and therefore the confidence intervals account for the ranges of both the total and the bin counts. We believe that this data would be useful for modeling Tor traffic distributions in tools like Shadow [19]. In the remainder of this section, we highlight some of the results.

We found that a majority of Web streams per circuit have fewer than 7 streams; by comparison, HTTP Archive [5] reports that 23% of pages have less than 10 connections per page. One potential reason for the relatively lower stream count may be because Tor Browser blocks javascript by default, which would prevent loading many embedded page objects. Most Other circuits have fewer than 3 streams.

We found that exit relays read less than 2 KiB and between 2 KiB and 16 KiB from 33% and 37% of Web streams, respectively, while they read less than 2 KiB from 56% of Other streams. Exit relays write less than 1 KiB to 77% of Web streams and 46% of Other streams. We also measured the "Bytes Per Stream Ratio" statistic that was computed using $log_2(bytes_{written}/bytes_{read})$ as input to the histogram for each stream. Under this interpretation, the range (-∞,-1) corresponds to the number of streams where the exit relay read more than twice as many bytes as wrote, the range [-1,1) corresponds to the number of streams where the exit relay read and wrote about the same amount, and the range [1,∞) corresponds to streams where the exit relay read less than half as much as wrote. Our results indicate that exit relays are mostly reading from Web streams, but that a bimodal distribution exists for Other streams where the exit relay writes more than reads for 12% of streams. Finally, we found that 68% of Web streams on the same circuit were created within 1 seconds of one another, while Other streams were more evenly dispersed across the bins we counted.

6. RELATED WORK

Privacy-Preserving Data Aggregation. There are many designs for privacy-preserving data aggregation in networks [6, 9, 15, 25]. The most relevant of these, and work that we build on directly, is the PrivEx system of Elahi *et al.* This system is designed specifically to gather statistics on Tor. PrivEx provides the S2 protocol, which we build on to create PrivCount, and the D2 protocol. The D2 protocol uses homomorphic encryption and zero-knowledge proofs to provide some additional security and tolerance against Tally Key Server (*i.e.*, PrivCount SK) faults. We choose to build on the simpler and faster S2 protocol because the security improvements of D2 are relatively minor, and we are able to add fault tolerance of the more error prone DCs to S2.

PrivCount expands on PrivEx-S2 in many ways to make it more flexible and practical. In general, the modifications make the protocol suitable for exploratory and diverse Tor

Table 11: Distributions of Tor network activity from histogram-counter in-depth exit statistics

Statistic		Bin Ranges and Count Distribution (with ± 95% CI)							
Active Circuit Life Time (s)		[1, 480):	57%±44%	[480, 720):	45%±42%	[720, 1200):	0%±33%	[1200, ∞):	0%±35%
Streams Per Circuit	Total	[1, 3):	46%±43%	[3, 7):	38%±41%	[7, 15):	31%±40%	[15, ∞):	9%±37%
	Web	[1, 3):	36%±37%	[3, 7):	22%±33%	[7, 15):	13%±31%	[15, ∞):	3%±28%
	Other	[1, 3):	78%±15%	[3, 7):	10%±9%	[7, 15):	0%±8%	[15, ∞):	2%±8%
Client-bound Bytes Per Stream	Total	[1, 2048):	60%±40%	[2048, 16384):	38%±35%	[16384, 65536):	32%±33%	[65536, ∞):	6%±26%
	Web	[1, 2048):	33%±33%	[2048, 16384):	37%±34%	[16384, 65536):	5%±26%	[65536, ∞):	0%±24%
	Other	[1, 2048):	56%±21%	[2048, 16384):	9%±15%	[16384, 65536):	8%±15%	[65536, ∞):	11%±15%
Server-bound Bytes Per Stream	Total	[1, 512):	57%±39%	[512, 1024):	25%±31%	[1024, 4096):	38%±34%	[4096, ∞):	0%±24%
	Web	[1, 512):	41%±35%	[512, 1024):	36%±34%	[1024, 4096):	23%±30%	[4096, ∞):	2%±25%
	Other	[1, 512):	40%±19%	[512, 1024):	6%±14%	[1024, 4096):	15%±16%	[4096, ∞):	1%±14%
Bytes Per Stream Ratio	Total	(-∞, -1):	80%±45%	[-1, 1):	25%±31%	[1, ∞):	0%±21%		
	Web	(-∞, -1):	70%±42%	[-1, 1):	15%±28%	[1, ∞):	0%±21%		
	Other	(-∞, -1):	45%±20%	[-1, 1):	14%±16%	[1, ∞):	12%±15%		
Inter-stream Creation Time (s)	Total	[0, 1):	87%±47%	[1, 5):	16%±29%	[5, 10):	1%±25%	[10, ∞):	0%±23%
	Web	[0, 1):	68%±41%	[1, 5):	8%±27%	[5, 10):	13%±28%	[10, ∞):	14%±28%
	Other	[0, 1):	16%±16%	[1, 5):	10%±15%	[5, 10):	3%±14%	[10, ∞):	12%±15%

measurements by a small deployment while maintaining its suitability for Tor-wide measurement. Major contributions of the PrivCount design over PrivEx include multi-phase iterative measurement, an expanded privacy notion that simultaneously handles multiple types of measurements, optimal allocation of the ϵ privacy budget across multiple statistics, and a composable security definition and proof. Our implementation is also a more capable and reliable tool, with 29 new Tor statistics, resilience against node failure and reboots, and simpler configuration and setup.

Measurement. There have been few published studies measuring Tor network traffic characteristics. Perhaps the most well-known is the 2008 study by McCoy *et al.* [23] in which the authors ran an exit relay and used `tcpdump` to capture the first 150 bytes of every packet (which included up to 96 bytes of application payloads). The study raised ethical and legal questions [28], as they collected, stored, and manually analyzed sensitive data. Chaabane *et al.* [8] conducted a similar study, but used a customized deep packet inspection software (OpenDPI) to further analyze packets by protocol.

Although we also measure Tor, privacy is a primary motivation of our work: PrivCount provides formal guarantees about security and privacy and the only outputs from the measurement process are the aggregated, noisy counts that are safe to share publicly. Further, to the best of our knowledge, neither of the previously mentioned studies used a non-default exit policy, while we found that a non-trivial shift in traffic type occurs when switching from the default exit policy to one that allows common file-sharing ports (Section 5).

In a case study on Tor measurement, Loesing *et al.* [22] measured countries of connecting clients and exiting traffic by port while providing guidelines for safely measuring potentially sensitive data in anonymity networks. They obfuscate the true measured values by rounding them to common multiples, but unlike PrivCount they do not aggregate across relays or provide any formal definitions of security or privacy. Tor currently allows relays to enable the collection and distribution [4] of statistics implemented by Loesing *et al.*, but it is not currently enabled by default.

Tor hidden service usage has also been measured more recently [16, 26]. We chose not to focus on hidden services because they account for less than 4% of all Tor traffic [20]. However, PrivCount provides a promising way to measure hidden-service statistics that cannot be safely gathered without aggregation, such as the number of hidden-service connections (see [16, Section 4]).

7. CONCLUSION

We build on recent advancements in privacy-preserving aggregation to develop a Tor measurement system called PrivCount. We detailed the PrivCount protocol and provided formal arguments about its security and privacy properties. We also implemented PrivCount and used it to perform a measurement study on Tor users and traffic. Among the measured statistics that we collected, we found that on average 710 thousand clients are connected to Tor at any given time, of which 550 thousand (77%) are active. Although Tor has its own estimates of the number of clients, we are the first to measure the distinct number of *active* clients. We also measured Tor traffic characteristics under various exit policies, providing the first analysis of how exit policies affect traffic distribution to the best of our knowledge. We found that Web ports account for 91% of bytes exiting Tor.

Future Work. Future work should consider taking measurements over longer time periods and with various exit policies in order to improve inference accuracy. Additional insights could be obtained with our tools and methods by extending the set of collected statistics to include onion service statistics. More robust and secure methods for counting the client population over varying time periods and without sacrificing privacy could allow for analysis of client churn. Finally, our measurements are well-suited to advance Tor research, especially in the area of traffic modeling and simulation.

Acknowledgments. We thank the anonymous reviewers for their suggestions and feedback which have helped to improve this paper. We also thank Tim Wilson-Brown, Tariq Elahi, David Goulet, and Micah Sherr for their assistance in running share keepers for our deployment. This work has been partially supported by the Defense Advanced Research Project Agency (DARPA), the National Science Foundation (NSF) under grant number CNS-1527401, and the Department of Homeland Security (DHS) Science and Technology Directorate, Homeland Security Advanced Research Projects Agency, Cyber Security Division under agreement number FTCY1500057. The views expressed in this work are strictly those of the authors and do not necessarily reflect the official policy or position of DARPA, NSF, DHS, or the U.S. Naval Research Laboratory.

8. REFERENCES

[1] PrivCount source code. https://github.com/privcount/privcount.

[2] TC: A Tor control protocol (Version 1). https://gitweb.torproject.org/torspec.git/tree/control-spec.txt.

[3] Tor directory protocol, version 3. https://gitweb.torproject.org/torspec.git/plain/dir-spec.txt.

[4] Tor Metrics. https://metrics.torproject.org/.

[5] HTTP Archive. http://httparchive.org/, May 2016.

[6] BURKHART, M., STRASSER, M., MANY, D., AND DIMITROPOULOS, X. A. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security Symposium* (2010).

[7] CANETTI, R. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science* (2001).

[8] CHAABANE, A., MANILS, P., AND KAAFAR, M. Digging into anonymous traffic: A deep analysis of the Tor anonymizing network. In *IEEE Network and System Security* (2010).

[9] CHEN, R., REZNICHENKO, A., FRANCIS, P., AND GEHRKE, J. Towards statistical queries over distributed private user data. In *USENIX NSDI* (2012).

[10] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *USENIX Security* (2004).

[11] DWORK, C. Differential privacy. In *International Colloquium on Automata, Languages and Programming* (2006).

[12] DWORK, C., KENTHAPADI, K., MCSHERRY, F., MIRONOV, I., AND NAOR, M. Our data, ourselves: Privacy via distributed noise generation. In *Eurocrypt*. 2006.

[13] DWORK, C., MCSHERRY, F., NISSIM, K., AND SMITH, A. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference* (2006).

[14] ELAHI, T., DANEZIS, G., AND GOLDBERG, I. Privex: Private collection of traffic statistics for anonymous communication networks. In *ACM Conference on Computer and Communications Security* (2014).

[15] ERLINGSSON, U., PIHUR, V., AND KOROLOVA, A. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *ACM Conference on Computer and Communications Security* (2014).

[16] GOULET, D., JOHNSON, A., KADIANAKIS, G., AND LOESING, K. Hidden-service statistics reported by relays. Tech. rep., The Tor Project, Inc., April 2015.

[17] HARDT, M., AND ROTH, A. Beating randomized response on incoherent matrices. In *ACM Symposium on Theory of Computing* (2012).

[18] JANSEN, R., BAUER, K., HOPPER, N., AND DINGLEDINE, R. Methodically modeling the Tor network. In *USENIX Conference on Cyber Security Experimentation and Test* (2012).

[19] JANSEN, R., AND HOPPER, N. Shadow: Running Tor in a box for accurate and efficient experimentation. In *Network and Distributed System Security Symposium* (2012).

[20] KADIANAKIS, G., AND LOESING, K. Extrapolating network totals from hidden-service statistics. Tech. rep., The Tor Project, January 2015.

[21] LINDELL, Y. *Composition of Secure Multi-Party Protocols, A Comprehensive Study*. Lecture Notes in Computer Science. Springer, 2003.

[22] LOESING, K., MURDOCH, S. J., AND DINGLEDINE, R. A case study on measuring statistical data in the Tor anonymity network. In *Financial Cryptograpy and Data Security* (2010).

[23] MCCOY, D., BAUER, K., GRUNWALD, D., KOHNO, T., AND SICKER, D. Shining light in dark places: Understanding the Tor network. In *Privacy Enhancing Technologies Symposium* (2008).

[24] MCSHERRY, F., AND MIRONOV, I. Differentially private recommender systems: building privacy into the net. In *ACM knowledge discovery and data mining (KDD)* (2009).

[25] MELIS, L., DANEZIS, G., AND CRISTOFARO, E. D. Efficient private statistics with succinct sketches. In *Network and Distributed System Security Symposium* (2015).

[26] OWEN, G., AND SAVAGE, N. Empirical analysis of Tor hidden services. *IET Information Security* (2016).

[27] SHIVA PRASAD KASIVISWANATHAN, A. S. On the 'semantics' of differential privacy: A bayesian formulation. *J. of Privacy and Confidentiality 6*, 1 (2014).

[28] SOGHOIAN, C. Enforced community standards for research on users of the Tor anonymity network. In *Workshop on Ethics in Computer Security Research* (2011).

APPENDIX

A. DETERMINING OPTIMAL NOISE

Given ϵ and δ for the statistics, PrivCount allocates ϵ and δ across the statistics by choosing ϵ_k and δ_k for each statistic s_k. Then, given ϵ_k and δ_k, it computes a value σ_k such that the Gaussian mechanism [12] with standard deviation σ_k provides ϵ_k-differential privacy with probability $1-\delta_k$. This guarantees (ϵ,δ)-differential privacy across all statistics. We first give an algorithm to compute σ_k, next give a counterexample showing that the formula of Elahi *et al.* [14] (Section 4.4.1) does not guarantee (ϵ,δ)-differential privacy, and finally give an algorithm to allocate ϵ to minimize across all statistics the maximum ratio of σ_k to the estimated value of the statistic v_k.

A.1 Computing $\sigma^*(\epsilon,\delta)$

Simple formulas to determine σ given ϵ and δ are presented for the single-dimensional case by Dwork et al. [12] (Section 2.1) and for the multi-dimensional case by Hardt and Roth [17] (Theorem 2.6). However, for the single-dimensional Gaussian mechanism, we can slightly improve upon this while still providing (ϵ,δ)-differential privacy across all statistics.

Let $\mathcal{D}$ be the set of possible inputs among which we have defined adjacency (see Section 2.3 for a discussion of adjacency). Let $D \sim D'$ indicate that inputs $D, D' \in \mathcal{D}$ are adjacent. Let mechanism $\mathcal{M} : \mathcal{D} \to R$ be a randomized algorithm with some output space R. The definition of (ϵ,δ)-differential privacy is as follows:

DEFINITION 1. *$\mathcal{M}$ satisfies (ϵ,δ)-differential privacy if, for all adjacent inputs D and D' and all $S \subseteq R$,*

$$Pr[\mathcal{M}(D) \in S] \le e^{\epsilon} Pr[\mathcal{M}(D') \in S] + \delta.$$

Let $f : \mathcal{D} \to \mathbb{R}$ be some query. The Gaussian mechanism depends on the *sensitivity* of f, which is defined as follows:

DEFINITION 2. *The sensitivity of f is*

$$\Delta_f = \max_{\substack{D\sim D':\\ D,D'\in\mathcal{D}}} |f(D) - f(D')|.$$

To define the Gaussian mechanism formally, recall from Section 2.2 that $\mathsf{Normal}(\mu,\sigma)$ denotes the normal distribution with mean μ and standard deviation σ. Given Δ_f, the Gaussian mechanism using the bound for σ of Dwork *et al.* [12] is as follows:

DEFINITION 3. *Let $\sigma(\epsilon,\delta) = \Delta_f \epsilon^{-1}\sqrt{2\ln(2/\delta)}$. Let $N \sim \mathsf{Normal}(0,\sigma(\epsilon,\delta))$. The Gaussian mechanism $\mathcal{G} : \mathbb{R} \to \mathbb{R}$ is*

$$\mathcal{G}(x) = f(x) + N.$$

$\mathcal{G}$ satisfies (ϵ,δ)-differential privacy for $\epsilon,\delta \le 1$ [12].

Recall (Section 2.2) that $\phi(\mu,\sigma;x)$ denotes the probability density function of the normal distribution with mean μ and standard deviation σ. Similarly, let $\Phi(\mu,\sigma;x)$ denote the cumulative distribution function of the normal distribution. The value $\sigma(\epsilon,\delta)$ is chosen such that the set of values x such that $\phi(0,\sigma;x) \le e^{\epsilon}\phi(\Delta_f,\sigma;x)$ has probability at least $1-\delta$. We can reduce the noise variance somewhat by finding the smallest value σ^* with this property. To compute σ^*, PrivCount simply performs binary search in $(0, \Delta_f\epsilon^{-1}\sqrt{2\ln(2/\delta)}]$ for the smallest value σ such that $\Phi(0,\sigma;x^*) \le \delta$, where $x^* = -(\epsilon\sigma^{*2}/\Delta_f) + \Delta_f/2$, which is the smallest value x satisfying $\phi(0,\sigma;x)/\phi(\Delta_f,\sigma;x) \le e^{\epsilon}$.

A.2 Counterexample for PrivEx σ

We show that Elahi *et al.* provide an invalid formula relating σ, ϵ, and δ (see [14], Section 4.4.1). They state that, after determining a σ that provides the desired limit on the adversary's "advantage", (ϵ,δ)-differential privacy holds for any ϵ and δ such that the following is satisfied:

$$\sigma \ge \frac{\Delta_f}{\epsilon}\sqrt{\ln\left(\frac{1.25}{\delta}\right)}.$$

However, the following counterexample shows that this is incorrect.

Suppose that $\epsilon = 0.2$, $\delta = 10^{-6}$, and $\Delta_f = 1$. Then set

$$\sigma = \frac{1}{0.2}\sqrt{\ln\left(\frac{1.25}{10^{-6}}\right)} \approx 18.734.$$

Then let

$$x^* = -\frac{\epsilon\sigma^2}{\Delta_f} + \frac{\Delta_f}{2} \approx -69.693.$$

Then $\Phi(0,\sigma;x^*) = 9.956\cdot 10^{-5}$, and $\Phi(\Delta_f,\sigma;x^*) = 8.048\cdot 10^{-5}$. Thus, $\Phi(0,\sigma;x^*) - e^{\epsilon}\Phi(\Delta_f,\sigma;x^*) = 1.257\cdot 10^{-6} > \delta$, which violates (ϵ,δ)-differential privacy.

A.3 Optimally allocating ϵ

Given $\delta_k = \delta/l$ for all l, PrivCount divides ϵ among the statistics in order to make their noise proportional to their expected values v_k. More precisely, it chooses $\epsilon_k > 0$ minimizing $\max_k \sqrt{n}\sigma_k/v_k$, where $\sigma_k = \sigma^*(\epsilon_k,\delta_k)$ and $\sum_k \epsilon_k = \epsilon$. This is equivalent to finding some global noise ratio ρ and ϵ_k such that $\sqrt{n}\sigma_k/v_k = \rho$, for all k, because otherwise the maximum ratio could be decreased be increasing the ϵ_k of the maximizing statistic and decreasing by the same amount the ϵ_k of some statistic with a smaller noise ratio.

Given some global ratio ρ, it must be that $\sigma_k = \rho v_k/\sqrt{n}$. Then each ϵ_k can be computed from σ_k and δ_k via binary search in a process very similar to that used to calculate σ_k from ϵ_k and δ_k. Such ρ is feasible if and only if $\sum_k \epsilon_k \le \epsilon$. Thus we can find the optimal allocation for ϵ by performing a binary search for the smallest ratio ρ that is feasible. Let ρ^* be this smallest ratio.

To determine a range in which to search for ρ^*, we begin by computing an optimal allocation using the upper bound of the values σ_k from Dwork *et al.* [12]:

$$\sigma'(\epsilon_k,\delta_k) = \Delta_k\epsilon_k^{-1}\sqrt{2\ln(2/\delta_k)},$$

where Δ_k is the sensitivity of statistic s_k. We can solve for the optimal allocation ϵ'_k using σ' for the σ_k by setting the noise ratios all equal:

$$\begin{aligned}
&\forall_k \frac{\sqrt{n}\sigma'(\epsilon'_1,\delta_1)}{v_1} = \frac{\sqrt{n}\sigma'(\epsilon'_k,\delta_k)}{v_k}\\
\Leftrightarrow\ &\forall_k \frac{\Delta_1\sqrt{2\ln(2/\delta_1)}}{\epsilon'_1 v_1} = \frac{\Delta_k\sqrt{2\ln(2/\delta_k)}}{\epsilon'_k v_k}\\
\Leftrightarrow\ &\forall_k \frac{\Delta_1}{\epsilon'_1 v_1} = \frac{\Delta_k}{\epsilon'_k v_k}\\
\Leftrightarrow\ &\forall_k \epsilon'_k = \frac{\Delta_k v_1 \epsilon'_1}{\Delta_1 v_k}\\
\Rightarrow\ &\epsilon'_1 = \frac{\epsilon}{1 + \sum_{k>1}(\Delta_k v_1)/(\Delta_1 v_k)},
\end{aligned}$$

where the last line holds because $\sum_k \epsilon'_k = \epsilon$.

Given this allocation, we can be certain that

$$\rho^* \in \left[\min_k \frac{\sqrt{n}\sigma^*(\epsilon'_k, \delta_k)}{v_k}, \max_k \frac{\sqrt{n}\sigma^*(\epsilon'_k, \delta_k)}{v_k}\right],$$

which is true for *any* allocation of ϵ because adjusting the given allocation in order to equalize the noise ratios can only decrease the maximum noise ratio and increase the minimum noise ratio. Using the σ' approximation as a basis for determining this range is particularly good to the extent that σ' is generally close to the optimal σ^*. In fact, if $\sigma' \leq c\sigma^*$ for some $c \geq 1$ (recall that we already have that $\sigma^* \leq \sigma'$), then computing ρ^* to within some accuracy ι takes at most $\log_2((1-1/c)\rho^*/\iota)$ rounds during the binary search. To see this, let $\rho' = (\sqrt{n}\sigma'(\epsilon'_k, \delta_k))/v_k$, which is well-defined because the ϵ'_k are set such that this ratio is equal for all k. Then observe that

$$\max_k \frac{\sqrt{n}\sigma^*(\epsilon'_k, \delta_k)}{v_k} \leq \rho'$$

because $\sigma^* \leq \sigma'$ and that

$$\min_k \frac{\sqrt{n}\sigma^*(\epsilon'_k, \delta_k)}{v_k} \geq \rho'/c$$

because $\sigma^* \geq \sigma'/c$. Finally, observe that $\rho' - \rho'/c \geq \rho^* - \rho^*/c$ because $\rho' \geq \rho^*$.

B. SECURITY AND PRIVACY PROOFS

THEOREM 1. *PrivCount UC-realizes $\mathcal{F}_{\mathsf{PC}}$ in the hybrid PKI model against any adversary that does not corrupt all SKs.*

PROOF. The simulator runs the PrivCount protocol internally with the given adversary $\mathcal{A}$. For honest parties, it receive the inputs up to the execution phase, and so it can simulate PrivCount in the first two phases. The PKI ensures authenticity of all messages in both the real and simulated protocols.

During broadcast, an abort message corresponds to a party receiving an inconsistent "rebroadcast" from another party. This event can only occur with a corrupt TS, and so the simulator can always observe it and can translate the event between the simulated protocol and the ideal world in either direction.

During aggregation, for each honest DC or SK P_i the simulator can use as counter values the initialization values that it sampled for the DCs. It can also use shared values created at honest DCs and SKs.

If the TS is corrupt, then after the last honest party P_i leaves execution, the simulator will receive the aggregated output of honest DCs y. It can than send as the final value to the TS from P_i to be to be value left after subtracting from y the sum of the initial counter values of all honest DCs. This is indistinguishable from a real PrivCount execution from the view of the adversary because (*i*) in both cases each value from a DC is independent and uniformly random by virtue of its shared value with any given one of the honest SKs, say, P_i; (*ii*) the value from each honest SK except P_i is independent and uniformly random by virtue of its shared value with any honest DC, and (*iii*) P_i is the one value such that the final sum is y plus any values shared between honest DCs and corrupt SKs and between honest SKs and corrupt DCs.

If the TS is not corrupt, then the simulator can extract the change in counter value $\Delta_{k,b}$ that adversary claims in its values to the TS by subtracting from the sum of final values from corrupt DCs and SKs the sum of the values shared between corrupt DCs and honest SKs and the initial values of the counters of corrupt DCs. Then the output value from the TS is the same in the real and ideal worlds.

Finally, suppose that a corrupt TS sends a subset $\mathcal{S}_i$ that doesn't contain a minimal DC set. Then the simulator receives no output from aggregation (just $\perp$), but it needs none because some SK never provides a final value to the TS. If instead a corrupt TS sends as inputs to two honest SKs subsets $\mathcal{S}_i$ that contain different honest DCs, then the simulator can send random final values to the TS from all DCs and SKs, because in the real PrivCount execution such subsets would also result in uniformly random final values as each SK contains a shared value not included in any other sum. $\square$

THEOREM 2. *If at least one SK $\mathcal{S}_i$ is honest, and if, for each minimal subset $\mathcal{S}$ of DCs in the deployment document that $\mathcal{S}_i$ receives with honest subset $\mathcal{H} \subseteq \mathcal{S}$, $\sqrt{\sum_{\mathcal{D}_i \in \mathcal{H}} w_i^2} \geq 1$, then the output of PrivCount is (ϵ, δ)-differentially private.*

PROOF. Because we have an honest SK, we can apply Thm. 1 and examine the output of $\mathcal{F}_{\mathsf{PC}}$. That functionality outputs to the environment or the adversary either something uncorrelated with the inputs or the sum of the initialized counters. The latter output has noise with standard deviation at least $\sqrt{\sum_{\mathcal{D}_i \in \mathcal{H}} w_i^2 \sigma_k} \geq \sigma_k$ in each counter. The values σ_k are calculated such that for each k the set of values x such that $\phi(0, \sigma_k; x) \leq e^{\epsilon_k}\phi(\Delta_k, \sigma_k; x)$ has probability at least $1 - \delta_k$, where Δ_k is the sensitivity of the kth statistic. Thus the set of outputs $x = (x_1, \ldots, x_l)$ such that $\phi(0, \sigma_k; x_k) > e^{\epsilon_k}\phi(\Delta_k, \sigma_k; x_k)$ for some k has probability at most $\sum_k \delta_k = \delta$. For all other output vectors x, $\phi(0, \sigma_k; x_k) \leq e^{\epsilon_k}\phi(\Delta_k, \sigma_k; x_k)$ for all k, and so the set of such vectors S has probability under any input D at most $e^{\sum_k \epsilon_k} = e^{\epsilon}$ times the probability of S under an adjacent input D'. Thus, the set of output statistics from $\mathcal{F}_{\mathsf{PC}}$ in a given collection period is collectively (ϵ, δ)-differentially private. Moreover, it is sufficient just to show that the functionality output from a single collection period is differentially private because adjacent inputs databases only differ in behavior during some period no longer than the reconfiguration time, and the functionality will not change its configuration and start new measurements without waiting for that length of time. $\square$