HisTorε: Differentially Private and Robust Statistics Collection for Tor

Akshaya Mani
Georgetown University
am3227@georgetown.edu

Micah Sherr
Georgetown University
msherr@cs.georgetown.edu

***Abstract*—A large volume of existing research attempts to understand who uses Tor and how the network is used (and misused). However, conducting measurements on the live Tor network, if done improperly, can endanger the security and anonymity of the millions of users who depend on the network to enhance their online privacy. Indeed, several existing measurement studies of Tor have been heavily criticized for unsafe research practices.**

Tor needs privacy-preserving methods of gathering statistics. The recently proposed PrivEx system demonstrates how data can be safely collected on Tor using techniques from differential privacy. However, as we demonstrate in this paper, the integrity of the statistics reported by PrivEx is brittle under realistic deployment conditions. An adversary who operates even a single relay in the volunteer-operated anonymity network can arbitrarily influence the result of PrivEx queries. We argue that a safe and *useful* data collection mechanism must provide both privacy and integrity protections.

This paper presents HisTorϵ, a privacy-preserving statistics collection scheme based on (ϵ, δ)-differential privacy that is robust against adversarial manipulation. We formalize the security guarantees of HisTorϵ and show using historical data from the Tor Project that HisTorϵ provides useful data collection and reporting with low bandwidth and processing overheads.

I. Introduction

Tor [9] is an anonymity network composed of approximately 6000 volunteer-operated relays with an estimated 1.75 million daily users [31]. Like the Internet, it is both a production network with live users and a research platform on which researchers experiment with new protocols and implementations.[1] Unlike the Internet, however, Tor has several characteristics that make it a particularly unfriendly platform for conducting (ethical) empirical studies:

[1]This dual-use is sometimes the source of conflict. To "minimize privacy risks while fostering a better understanding of the Tor network and its users", the Tor Project recently established a Tor Research Safety Board [32] that maintains guidelines and offers feedback to researchers concerning the potential risks of experimenting on the live Tor Network.

NDSS '17, 26 February - 1 March 2017, San Diego, CA, USA
Copyright 2017 Internet Society, ISBN 1-891562-46-0
http://dx.doi.org/10.14722/ndss.2017.23411

These characteristics include: (1) *dynamics*—Tor is a highly dynamic network that is known to be affected by world events. For example, uprisings in undemocratic nations tend to be followed by more advanced attempts to block Tor from within those countries [31]. Entire countries are periodically blocked and then allowed to access the network. Comprehensive statistics gathering efforts must consider both short-term and more longitudinal trends; (2) *privacy*—Tor is designed to protect the privacy of its users. Experiments that capture users' communications are thus antithetical to the goals of the anonymity network and can potentially endanger users who depend on the network to hide their identities; (3) *integrity*—unlike the Internet, the core of Tor's network is operated by volunteers. It is far easier to operate a malicious Tor relay than a core Internet router, and hence measurement studies should consider a threat model in which the adversary attempts to manipulate measurements to further its goals; and (4) *security*—similarly, a privacy-preserving measurement system should not provide a new attack surface for disrupting or otherwise manipulating Tor.

We posit that the above challenges contribute to the lack of understanding of actual Tor users. We know surprisingly little about who uses the network, what they are using it for, and how they are using it. We discuss the lack of information about how people are using Tor in more detail in the next section.

While our lack of knowledge about how Tor is used in practice may at first blush signal a strength of Tor (that is, that it successfully conceals its users' behavior), it also limits our ability to analyze the actual privacy properties of the network. For example, attempts to empirically measure the anonymity offered by the Tor network are predicated on maintaining *accurate* models of real Tor users' behavior [2, 14, 17, 19, 33]. To date, these models have been largely best guesses.

The core contribution of this paper is the introduction of HisTorϵ, a differential privacy scheme for Tor that is scalable, incurs low overheads, preserves users' privacy, and provides strong integrity guarantees[2]. The goal of HisTorϵ is to provide researchers with a platform for conducting accurate measurement studies on Tor without endangering the network's users and while incurring only low overheads.

Importantly, HisTorϵ is not the first differentially private statistics gathering scheme for Tor. Elahi et al. [15] recently proposed a differential privacy solution for Tor called PrivEx. PrivEx has significantly raised the bar for safe Tor measurements. Indeed, the Tor Research Safety Board cites PrivEx

[2]HisTorϵ is pronounced as "history."

as an example of exciting research towards conducting *safe* measurements on the live network [32].

A key distinguishing feature of HisTorϵ is its robustness to manipulation. PrivEx raises the bar on Tor experimentation by providing privacy. We show, however, that PrivEx is not immune to manipulation. In particular, we demonstrate that a malicious relay operator can drive the aggregate statistic being calculated towards an arbitrary value of its choosing. While such an attack may be outside of the threat model envisioned by Elahi et al., the attack points to the need for statistics gathering mechanisms that are both privacy-preserving and resilient to manipulation.

HisTorϵ's robustness is mostly achieved by forgoing general counting queries (as supported by PrivEx) in favor of supporting only *binning* queries. Although we support more general binning, a useful example of the type of queries that HisTorϵ supports is a histogram. In HisTorϵ, each data contributor (for example, exit relays) must contribute either "0" or "1" to the total count in a bin. For instance, an analyst can query HisTorϵ to provide a histogram that shows the number of connections observed by the different guard relays, or the number of connections to Hidden Service Rendezvous Points as observed by relays, etc. To provide robustness guarantees, HisTorϵ uses Goldwasser-Micali (GM) encryption [16] to ensure that encrypted values are either treated as 0 or 1, restricting the influence of a malicious relay. Put another way, all relays (malicious or not) are strictly bounded in their influence over the total aggregate. In the sections that follow, we demonstrate both analytically and empirically that HisTorϵ's binning strategy supports a wide range of queries useful for Tor researchers while enforcing strong integrity guarantees and imposing low communication and computation overheads.

HisTorϵ also provides resistance to so called "compulsion attacks" in which a relay operator is compelled to release information to a (potentially totalitarian) government. Statistics gathering requires, obviously, gathering statistics, which itself poses a privacy risk since these statistics would not otherwise be collected. For example, Tor does not currently log client connections to guards; queries which ask for a histogram of the number of guard connections thus may impose additional privacy risks. We design HisTorϵ to mitigate such risks through the use of encrypted data structures we call *oblivious bin counters*. Simply put, we use efficient cryptographic techniques and data structures to maintain counters that the relay operators cannot read on their own.

II. Background

Tor. Tor is a network of volunteer-operated routers that enhances privacy by separating network identity from network location [9]. The Tor client proxies a user's TCP connections through a series of randomly selected *relays*, which collectively form a *circuit*. Several TCP connections may be multiplexed over the same circuit. The first hop on this circuit—the ingress point to the Tor network—is usually a Tor *guard* relay, a Tor relay that is relatively stable and is deemed to have sufficient bandwidth for the task. As a notable exception, traffic may also enter the Tor network through bridge nodes, which are similar to guards but are not publicly advertised. Traffic exits the Tor network through exit relays, which establish TCP connections to the intended destination. Along the Tor circuit, cryptography helps conceal the actual sender and receiver—relays know only the previous and next hop along the anonymous path.

Tor is designed to be robust against a non-global adversary, meaning that it provides protections against an adversary who cannot view all traffic on the network. It is known to be vulnerable against *traffic correlation* attacks [20] in which an adversary can observe an anonymous connection as it enters and leaves the anonymity network. Here, using timing and flow analysis, the adversary can correlate the ingress and egress traffic and determine that they belong to the same traffic stream, thus de-anonymizing both endpoints of the communication.

Our current understanding of Tor's usage is limited. The information we do have about Tor's real-world usage is unfortunately incomplete, outdated, and sometimes even contradictory.

The study by McCoy et al. [24] is perhaps the earliest attempt at understanding how Tor is used, but its findings are now almost a decade old and reflect a historical Tor network that had one fifth of the relays and fewer than half of the number of daily users as the network does today [31]. More importantly, the surreptitious capturing and analysis of real users' anonymized traffic flows is now viewed as ethically ambiguous and even potentially harmful; indeed, the work is sometimes cited as being an exemplar of unsafe security research [30].

The ongoing debate [28] between supporters of the Tor network and the CloudFare CDN further highlights the lack of a clear understanding of how Tor is used. Briefly, CloudFare forced Tor users to complete user-unfriendly CAPTCHAs before accessing any of the vast number of CloudFare's hosted sites. The CTO of CloudFare cited the large portion of attack traffic originating from Tor exit relays as the principle motivation of the CAPTCHAs. The Tor Project publicly responded [27] by questioning the accuracy of CloudFare's measurements, and pointed to a report by Akamai [1] showing that Tor traffic had a similar proportion of attack traffic as the regular (non-anonymized) Internet.

Similarly, researchers have published measurement studies that show that an enormous percentage of connections to Tor Hidden Services go to hidden service sites that serve child pornography [4]. This is again disputed by the maintainers of the Tor Project [23].

Background on differential privacy. Differential privacy [10] seeks to minimize the privacy risk of participating in a database while maximizing the accuracy of statistical queries against that database. Although several notions of differential privacy exist, this paper considers (ϵ, δ)-differential privacy as introduced by Dwork et al. [11]: a computation $\mathcal{F}$ gives (ϵ, δ)-differential privacy if, for all data sets D_1 and D_2 that differ on only one row, and all $S \subseteq \text{Range}(\mathcal{F})$,

$$\Pr[\mathcal{F}(D_1) \in S] \leq \exp(\epsilon) \times \Pr[\mathcal{F}(D_2) \in S] + \delta$$

Crucially, differential privacy's formal guarantees are about the properties of the database mechanism rather than the computational capabilities or auxiliary knowledge of the adversary. In practice, differential privacy is achieved by adding noise to the result of some aggregate query. The parameter ϵ controls the tradeoff between privacy and utility: a larger ϵ provides weaker privacy but more accurate results. The inclusion of δ relaxes the more rigid notion of ϵ-differential privacy and allows for more practical implementations.

In this paper, we adopt the distributed (ϵ, δ)-differential privacy scheme of Chen et al. [6]. In their work, an *analyst* poses a question to a fixed set of c *clients*. Each client c_i contributes a bit vector[3] $\mathbf{v}_i = \{v_{i,1}, \ldots, v_{i,q}\}$, encrypted using Goldwasser-Micali (GM) bit cryptosystem [16] with the analyst's public key. GM encryption enforces the property that ciphertext can only encode `0` or `1`. To provide privacy, a *proxy* adds in n encrypted random noise vectors, where n is calculated as

$$n = \lfloor \frac{64 \ln(2/\delta)}{\epsilon^2} \rfloor + 1 \tag{1}$$

The GM-encrypted $c + n$ client and noise vectors are then shuffled by the proxy and returned to the analyst. The analyst uses its private key to decrypt the bit vectors. Letting $d_{i,j}$ be the jth element of vector i (where $1 \le i \le c+n$), the analyst obtains the (noised) aggregate result a_j as

$$\mathbf{a}_j = \sum_{k=1}^{c+n} d_{j,k} - \frac{n}{2} \tag{2}$$

for all $1 \le j \le q$ (the number of elements in a bit vector) [6].

In the work by Chen et al. [6], the authors set $\delta < 1/c$ in Eq. 1. However, Dwork et al. [13] prove that such a value of δ is very dangerous, as it permits compromising the privacy of "just a few" number of clients. Therefore, we set δ to a much lower value, typically on the order of $10^{-6}/c$.

Also, we use $\epsilon = 1$ for our experiments, unless otherwise noted. We note that this offers more privacy than the experimental setting ($\epsilon = 5$) used by Chen et al. [6].

III. Manipulating PrivEx

PrivEx [15] is the first system for private data collection on the live Tor network. It has (justifiably) garnered significant attention from the privacy-enhancing technologies community and has been promoted as an example of a technology that enables safe data collection on Tor [32].

In comparison with HisTorϵ, which we introduce in the next section, PrivEx has a similar system model, but differs in that relays individually contribute their own noise. This has the advantage that malicious proxies cannot break privacy guarantees by not following the protocol. However, PrivEx makes implicit trust assumptions that allow *any* malicious relay to manipulate the results of a differentially private query.

Briefly, Elahi et al. introduce two PrivEx variants for private collection of traffic statistics on Tor. In both their secret-sharing and distributed decryption schemes, an invariant is that relays contribute their own individual value, and the system computes the sum of those values. That is, PrivEx supports summation queries. Importantly, there is no protection against an attack on the integrity of the statistics. This enables a malicious contributor (e.g., a relay) to submit an arbitrary value that can significantly impact the aggregate result.[4]

[3]We use a bold typeface to represent vectors.

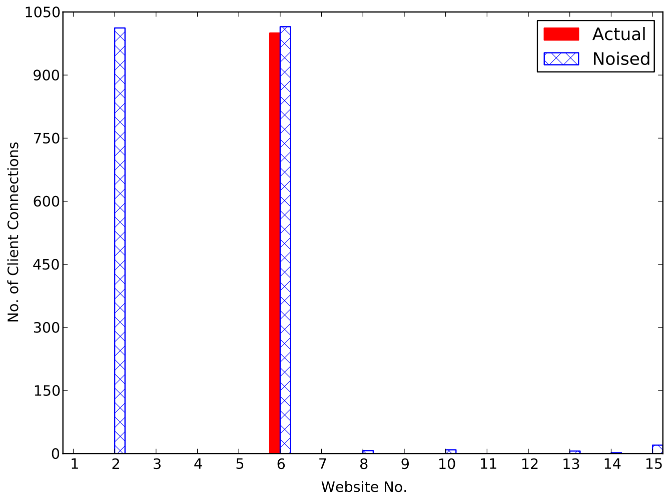

Fig. 1. The number of client connections (y-axis) for 15 different websites (x-axis) as actually occurred ("Actual") and as reported by PrivEx ("Noised"). A single malicious relay falsely reports that it observed 1000 connections to website #2.

To demonstrate this type of manipulation, we made a trivial modification to the PrivEx source code (as provided by Elahi et al. [15]) to include a single malicious relay. Consistent with their work, we consider a query that aggregates the number of visits destined to 15 particular websites. Such a query could be useful to answer the question: *which websites are most popular among Tor users?* In our experiment, we use the same default parameters as is included in the PrivEx source code. However, we modify the behavior of a single relay to falsely claim that it has observed 1000 visits to website #2.

Figure 1 shows the result of this manipulation. The actual distribution of website visits is shown in red, and indicates that website #6 is, by far, the most popular site visited via Tor. The noised distribution as reported by PrivEx, shown in patterned blue, paints a different picture: according to the results of the query, website #2 appears to similarly be popular amongst Tor users.

The above example is of course one of many possible manipulations. More generally, a single malicious relay that participates in PrivEx can supply any (false) count to manipulate the aggregated result, and do so without risking detection.

We emphasize that the above "attack" falls entirely outside of the threat model considered by Elahi et al. [15]. That is, we are not disclosing errors in their design or implementation.

This paper makes the argument that while PrivEx provides strong privacy guarantees, its lack of integrity guarantees is problematic for many types of queries that would be useful for Tor. Since Tor is a volunteer-operated network, the barrier to entry for operating a relay is very small. Indeed, there have been several instances [34] in which relay operators have been found to behave maliciously. While we view PrivEx as

[4]Elahi et al. posit that range proofs could potentially be added to PrivEx to bound the impact of untrue statistics. However, as the authors admit, such techniques lead to significant computation and communication overheads [15].

a significant step forward in private data collection, we argue in this paper that privacy must be combined with integrity protections to provide a useful statistics gathering service.

IV. HisTorϵ Overview

HisTorϵ's goal is to provide differentially private data collection for Tor that is robust against manipulation. We introduce HisTorϵ by describing its participants and system model (§IV-A), threat model (§IV-B), query capabilities (§IV-C), and operation (§IV-D).

A. Participants and System Model

There are three types of participants in HisTorϵ:

Data collectors (DCs) [15] are relays that collect statistics that will later be aggregated. Example of DCs and the data that they collect include guard relays that count the number of client connections, middle relays that count requests to Tor Hidden Service introduction points, and exit relays that keep track of exit bandwidth. Developing a comprehensive list of possible queries is beyond the scope of this paper. Our aim is rather to provide a secure and private statistics gathering technique for Tor that is sufficiently general to be adapted for many types of queries.

As with vanilla Tor, HisTorϵ relies on secure communication via TLS, and we use Tor's existing directory infrastructure as a trust anchor to locate public keys and authenticate messages. Since relays are already heavily burdened in Tor, we aim to keep both the computation and communication overheads of HisTorϵ low for the DCs.

The **analyst** is a party that issues queries to the data collectors and receives noised query responses. We envision that, at least initially, the analyst will be the maintainers of Tor.

Finally, we introduce three **mixes** that are third parties that provide the required amount of differentially private noise. Mixes are dedicated servers responsible for enabling HisTorϵ queries. We assume that all parties can obtain the mixes' public keys, which for example, could be attested to by the Tor directory authorities.

Also included in the HisTorϵ ecosystem are the Tor users who use the Tor client software to communicate via Tor, the destinations that the users are visiting, and the Tor directory servers and mirrors. Since HisTorϵ imposes no changes to the normal operations of Tor, we mostly omit discussing these components unless they are pertinent to the security and privacy properties of HisTorϵ.

B. Threat Model and (Informal) Security Guarantees

We consider both internal and external threats to privacy and data integrity:

Internal adversaries. DCs are volunteer-operated relays and can be malicious. A malicious DC is a Byzantine adversary that can, for example, disobey HisTorϵ protocols, submit false statistics, and/or refuse to participate. Malicious DCs may also collect and leak sensitive information (e.g., the IP addresses of clients or destination addresses); such leakage is also possible with existing Tor relays, and we do not consider defenses against such behavior here. HisTorϵ ensures that no colluding group of DCs can reveal more information than would otherwise be available by pooling their knowledge.

Malicious DCs may attempt to manipulate query results by reporting erroneous data, as in §III. If c is the number of DCs that participate in a query and f is the fraction of the participating DCs that are malicious, then HisTorϵ guarantees that the maximum influence over the aggregate result is $\pm fc$. This is a direct consequence of applying the (ϵ, δ)-differential privacy scheme of Chen et al. [6]: each relay (malicious or not) can contribute at most `1` to each element in its supplied vector. If a DC refuses to participate in a query or submits a malformed vector, its bit vector is considered to be all `0`s. Consequently, malicious DCs cannot disrupt (i.e., cause denial-of-service) HisTorϵ queries by submitting false or malformed data.

Malicious mixes may also behave arbitrarily. HisTorϵ provides strong privacy guarantees when (1) no more than one of the mixes is malicious and (2) a malicious mix does not collude with a malicious analyst. HisTorϵ employs secret sharing techniques to ensure that non-colluding mixes cannot learn un-noised query results.

Mixes can attempt to manipulate query results by adding false values, improperly constructing noise vectors, or modifying or discarding the encrypted vectors it receives from the DCs. HisTorϵ's integrity guarantees ensure that the analyst can detect manipulated query results as long as one of the mixes is honest.

In HisTorϵ, the analyst issues queries and receives noised results from the mixes. We consider a malicious analyst that colludes with other parties to attempt to learn non-noised answers to queries. HisTorϵ achieves (ϵ, δ)-differential privacy when no more than one mix is malicious and no malicious mix colludes with the analyst. If a malicious DC colludes with either a mix or the analyst, no information that is not already available to the malicious DC is revealed.

External adversaries. HisTorϵ is robust against an external adversary that observes all HisTorϵ-related communication. All HisTorϵ exchanges are secured by TLS. We assume that public keys can be reliably retrieved—for example, by leveraging Tor's existing directory infrastructure—and that all properly signed messages can be authenticated.

We also consider an external adversary that applies pressure (e.g., through a subpoena or threat of violence) to an honest DC to reveal its individual counters. We argue that papers (such as this one) that propose collecting information that would otherwise not be gathered by anonymizing relays have a responsibility to evaluate such *compulsion attacks*. We describe in §V techniques that limit the amount of information that can be "handed over" by a pressured honest relay to an adversary.

C. Queries

In HisTorϵ, for each query, each DC i contributes a bit vector $\mathbf{v_i}$ of length b, where b is a parameter of the query. For ease of exposition, we refer to each vector position as a *bin*.

The result of the query is the noised aggregate vector described in Eq. 2. Essentially, the query returns a vector of summations $\sum_{i=1}^{c} \mathbf{v_i}$ plus the added noise. The data and noise are shuffled to provide indistinguishability.

HisTorϵ supports two types of queries: class queries and histogram queries.

Class queries. In a class query, each bin j is assigned a class label C_j. For Tor, potentially useful class labels include (but are not limited to) protocols/ports seen by exit relays and client version numbers seen by guards. The semantics of class queries allow the analyst to ask for all $j \in [1, b]$: *how many relays have witnessed the event described by the label C_j?*

The analyst specifies the semantic meaning for each bin. For example, in the case of reporting which ports have been seen by exit relays, the analyst can specify a query such that the first bin in DCs' bit vectors indicates whether the exit has witnessed http traffic, the second bin indicates whether it has seen ssh traffic, etc. By examining the aggregated and noised results, the analyst learns approximately how many exit relays have seen the specified traffic types.

Histogram queries. Histogram queries allow the analyst to learn the distribution of some value, taken over the DCs. As examples, histogram queries can inform the analyst of the distribution of client connections seen by guards or the bandwidth seen by exit relays.

For histogram queries, each DC maintains an encrypted counter of the relevant statistic (e.g., observed bandwidth). Each bin j is assigned an interval $[L_j, U_j)$ where $L_j, U_j \in \mathbb{Z}$ such that $L_j \geq U_{j-1}$ when $j > 1$. When a bin $b_j = 1$, this indicates that the encrypted counter is in the range $[L_j, U_j)$. Note that at most one bin belonging to a DC is set to `1`; all other bins are set to `0`. (This is unlike a class query; there, multiple bins/classes can be set to `1`.)

The vector sum over all DCs' bin vectors (i.e., $\sum \mathbf{v_i}$) yields the distribution of the DCs' counters. As explained next, mixes add random noise vectors to apply differential privacy to this distribution.

D. Operation

HisTorϵ operates in loosely synchronized hour-long epochs. At the beginning of each epoch, each DC zeroes all of its bins. As with PrivEx, HisTorϵ ensures that data sets do not carry over between queries; that is, no more than one query within an epoch can cover a given bin. Extending HisTorϵ to support differential privacy when data must be continually observed [5, 12] is left as an interesting area of future work. Currently, as with PrivEx, we enforce independence between different epochs by zeroing all counters. (We include discussions of the absoluteness of this independence and the effect of a privacy budget for differentially private queries in §XI.)

HisTorϵ's workflow begins when the analyst issues a query to the DCs and mixes. Currently, this is manual process that requires configuring Tor relays to report the statistic of interest (e.g., connection counts, observed bandwidth, etc.). We envision that future versions of HisTorϵ will support SQL-like semantics to automate this process. Our prototype implementation, described in more detail in §IX, uses Tor's existing statistics module and can be easily configured to aggregate the data that Tor already collects. We have already added hooks for bandwidth and connection counting.

Figure 2 shows how queries are processed in HisTorϵ. DCs maintain three redundant copies of encrypted counters (encrypted binary vectors). Each copy is encrypted using the public key of one of the three mixes.

At the end of the epoch, these encrypted vectors are further obfuscated by xor'ing with a random binary vector R. Xor'ing with the random vector ensures that mixes cannot learn the plaintext of the DCs' binary vectors, even after decrypting with their private GM key. (Recall that GM is a homomorphic cryptosystem with respect to xor [16].) The DCs send one copy of its GM-encrypted and xor'd vector to each mix, as shown in Figure 2. Additionally, the DCs communicate secret shares of R across the three mixes.

Each mix then decrypts the GM encryption, yielding the original binary vectors xor'd with a random vector. Each such vector is added to a matrix; that is, this matrix contains the xor-encrypted vectors from the DCs. Mixes then add n randomly generated rows to the vector, where n is computed according to Eq. 1 (where, $\delta = 10^{-6}/c$). Finally, the vector is randomly shuffled columnwise. Both the addition of the n rows of noise and the shuffling is performed using cryptographically secure random seeds, which are shared amongst the mixes. Consequently, the three mixes add identical noise vectors and perform the identical shuffle.

Finally, the mixes communicate the resultant matrices to the analyst as well as the shuffled secret shares of the R random vector. The analyst then combines the shares to obtain the shuffled R, and uses it to decrypt both the shuffled data and noise records (which are indistinguishable) in the matrix. To obtain the aggregate, the analyst then subtracts the expected noise value according to Eq. 2.

V. Oblivious Counters

To mitigate compulsion attacks, HisTorϵ minimizes the amount of information that DCs need to maintain through the use of *oblivious counters*. In this section, we assume that the DCs are honest. A malicious DC (relay) need not use oblivious counters and can trivially leak sensitive statistics, regardless of whether HisTorϵ is used.

Each DC maintains three binary vectors of length b, where b is determined by the query. Each binary element of the first binary vector is GM-encrypted using the public key of the first mix; the second vector is GM-encrypted using the public key of the second mix, and so on. For ease of notation, we focus below on one such GM-encrypted vector which we denote as $\overset{e}{\mathbf{v}} = \langle E_+(v_1), \ldots, E_+(v_b) \rangle$, where $E_+(\cdot)$ denotes encryption using the public key belonging to the pertinent mix. The scheme is identical for the three encrypted binary vectors maintained by each DC.

A. Oblivious class counters

Class queries allow the analyst to discover how many DCs encountered an event (see §IV-C). Recall that each v_j corresponds to a class label C_j, which for example could denote a particular protocol or type of event. When $v_j = 1$,

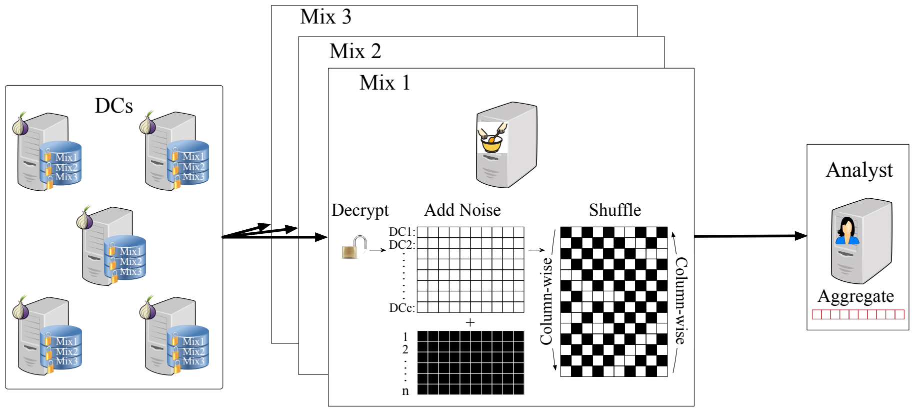

Fig. 2. An overview of HisTorϵ query processing. Each mix receives an encrypted vector from each DC. The mixes then add noise and perform a column-wise shuffle. The resulting vectors are then sent to the analyst, who in turn can compute the aggregate result.

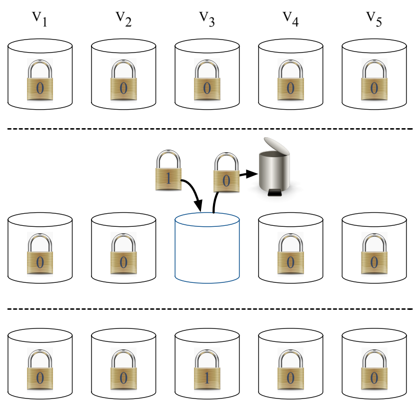

Fig. 3. Maintaining an oblivious class counter. *Top:* The counters ($b = 5$) are initialized to GM-encryptions of $\mathtt{0}$. *Middle:* An event corresponding to class C_3 is observed, and $E_+(v_3)$ is replaced with $E_+(\mathtt{1})$. *Bottom:* The resulting encrypted binary vector.

we say that the event has been observed by the DC; otherwise $v_j = 0$ indicates that the event was not witnessed.

At the beginning of each epoch, each DC zeroes its vector $\overset{e}{\mathbf{v}}$ by GM encrypting the bit $\mathtt{0}$, b times. Since the GM cryptosystem is a probablistic cryptosystem [16], we have that $\forall x, y \in [1, b]$, $x \neq y \Rightarrow E_+(v_x) \neq E_+(v_y)$ with high probability. That is, without knowledge of the corresponding private key, GM encryption guarantees that an adversary cannot determine whether $v_x \overset{?}{=} v_y$ when $x \neq y$.

When a DC observes an event corresponding to a class label C_j, it replaces the value of $E_+(v_j)$ in $\overset{e}{\mathbf{v}}$ with a new GM encryption of the bit $\mathtt{1}$. Note that if v_j was already $\mathtt{1}$, then the operation effectively replaces the old encryption of $\mathtt{1}$ with a new encryption of $\mathtt{1}$. This process is depicted in Figure 3.

Crucially, DCs do not store the plaintext vector elements $v_1, \ldots, v_b$, and instead maintain only the encrypted binary values $E_+(v_1), \ldots, E_+(v_b)$. Since the DCs also do not have the private key for decrypting the elements of $\overset{e}{\mathbf{v}}$, it trivially holds that an honest DC that does not know either v_j or the mix's private key cannot provide v_j to an adversary. That is, the above scheme trivially achieves resistence to the compulsion attack.

B. *Oblivious histogram counters*

In the case of histogram queries, recall that the analyst's query maps each vector element v_j to a range $[L_j, U_j)$ such that $L_j \geq U_{j-1}$ when $j > 1$. When v_j is $\mathtt{1}$, this indicates that the statistic of interest as measured by the DC is in the range $[L_j, U_j)$. Hence, at most one v_j is set to $\mathtt{1}$. As a special case, we set $U_b = \infty$.

Let w_j be the *bin width* of vector element v_j; i.e., $w_j = U_j - L_j$. We do not require that $\forall x, y \in [1, b], w_x = w_y$, but we denote this special case as *equal width* bins.

As explained below, our oblivious histogram counter scheme requires equal width bins. Since we do not require that $w_x = w_y$ for all $x, y \in [1, b]$—that is, the histogram query need not use equal width bins—we use an auxiliary binary vector with equal width bins, where the bin width is set to the greatest common divisor (GCD) of $w_1, \ldots, w_b$. More formally, we define $\overset{e}{\boldsymbol{\nu}} = \langle E_+(\nu_1), \ldots, E_+(\nu_\beta)\rangle$, where each element ν_j covers the range $[(j-1)g, jg)$ and g is the GCD of $w_1, \ldots, w_b$. As a special case, ν_β covers the range $[(\beta - 1)g, \infty)$. In practice, to reduce the size of $\overset{e}{\boldsymbol{\nu}}$, we carefully choose bin widths such that $\beta < 15000$. In the special case that the histogram query species equal width bins, we have that $b = \beta$ and $\overset{e}{\mathbf{v}} = \overset{e}{\boldsymbol{\nu}}$.

At the beginning of each epoch, each DC initializes its encrypted vector as $\overset{e}{\boldsymbol{\nu}} = \langle E_+(\mathtt{1}), E_+(\mathtt{0}), \ldots, E_+(\mathtt{0})\rangle$ (all but the first element are encryptions of $\mathtt{0}$). It also maintains a counter t, initialized to 0.

When the DC observes the statistic of interest (number of connections, a KB of consumed bandwidth[5], etc.), it executes the IncrHistCounter procedure.

The IncrHistCounter procedure works by shifting the position of the encrypted $\mathtt{1}$ in the $\overset{e}{\boldsymbol{\nu}}$ vector whenever the counter

[5]IncrHistCounter can be easily adapted for statistics that increase in increments greater than 1 (e.g., observed bandwidth). In particular, if k is the increase in the statistic, then $\overset{e}{\boldsymbol{\nu}}$ is right shifted $\lfloor (t + k)/g \rfloor$ positions and t is reset to $(t + k) \bmod g$.

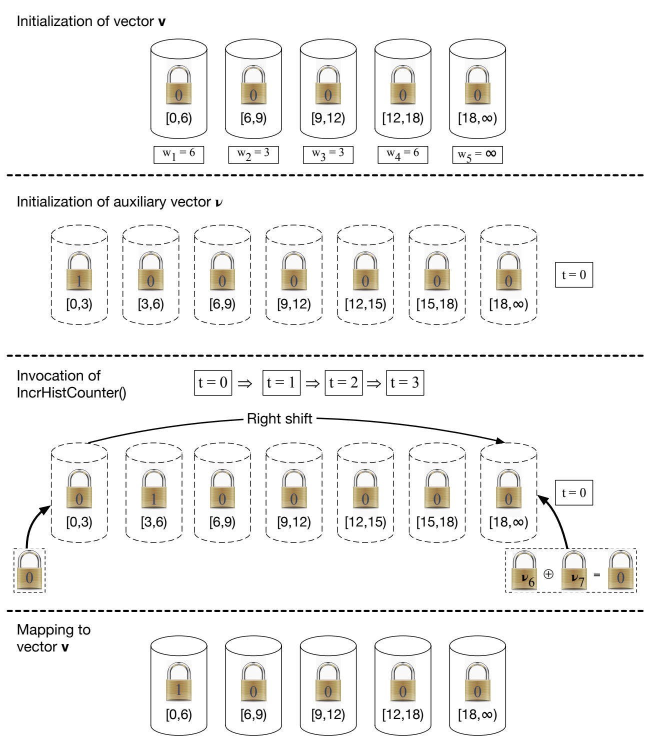

Fig. 4. Maintaining an oblivious histogram counter. *Top:* The analyst defines bin widths for $\overset{e}{\mathbf{v}}$. The GCD (g) is 3. *Second from top:* The initialization of $\overset{e}{\boldsymbol{\nu}}$. *Second from bottom:* After three observations ($t = 3$), the bins in $\overset{e}{\boldsymbol{\nu}}$ are right shifted. *Bottom:* At the end of the epoch, the values of $\overset{e}{\boldsymbol{\nu}}$ are mapped to $\overset{e}{\mathbf{v}}$.

```
1 $t \leftarrow t + 1$
2 if $t = g$                          // $g$ is GCD and bin width
3 then
4     $tmp \leftarrow E_+(\nu_{\beta-1}) \oplus E_+(\nu_\beta)$
5     $\overset{e}{\boldsymbol{\nu}} \leftarrow \overset{e}{\boldsymbol{\nu}} \gg 1$              // Right shift, no wrap
6     $E_+(\nu_\beta) \leftarrow tmp$           // GM is xor homomorphic
7     $E_+(\nu_1) \leftarrow E_+(0)$
8     $t \leftarrow 0$
9 end
```
Procedure IncrHistCounter

reaches the bin width. Line 6 handles the special "overflow" case: when the last bin is set to a 1, it always retains that 1 (recall that the last bin represents the range $[(\beta-1)g, \infty)$). An example invocation of IncrHistCounter is shown in Figure 4.

Since an honest DC does not maintain the plaintext values of $\nu_1, \ldots, \nu_\beta$ and does not know the decryption key, it cannot reveal which bin contains the 1, even if compelled to do so. Importantly, unlike oblivious class counters, oblivious histogram counters leak information—in particular, the counter t. More formally, we leak $\lfloor \log_2 g \rfloor + 1$ least significant bits of the DC's measured value, $(\sum_{v_i \in \mathbf{v}} v_i \cdot L_i) + t$, when a compulsion attack takes place (since $t \in [0, g)$).

At the end of the epoch, the encrypted values of $\overset{e}{\boldsymbol{\nu}}$ need to be mapped back to $\overset{e}{\mathbf{v}}$. Since the width of the ranges covered by $\overset{e}{\boldsymbol{\nu}}$ are defined by the GCD of the ranges covered by $\overset{e}{\mathbf{v}}$, it holds that each $E_+(\nu_j) \in \overset{e}{\boldsymbol{\nu}}$ maps to a single $E_+(v_k) \in \overset{e}{\mathbf{v}}$. For example, in Figure 4, we have $E_+(\nu_1) \mapsto E_+(v_1)$, $E_+(\nu_2) \mapsto E_+(v_1)$, $E_+(\nu_3) \mapsto E_+(v_2)$, $E_+(\nu_4) \mapsto E_+(v_3)$, $E_+(\nu_5) \mapsto E_+(v_4)$, $E_+(\nu_6) \mapsto E_+(v_4)$, and $E_+(\nu_7) \mapsto E_+(v_5)$, where $\mapsto$ signifies the mapping between $\overset{e}{\boldsymbol{\nu}}$ and $\overset{e}{\mathbf{v}}$. Let $\mathcal{M}(v_k, \overset{e}{\boldsymbol{\nu}})$ denote the set of elements of $\overset{e}{\boldsymbol{\nu}}$ that map to a given $v_k \in \overset{e}{\mathbf{v}}$. We can therefore compute

$$E_+(v_k) = \bigoplus_{E_+(\nu_j) \in \mathcal{M}(v_k, \overset{e}{\boldsymbol{\nu}})} E_+(\nu_j) \tag{3}$$

Eq. 3 holds since at most at one element in $\mathcal{M}(v_k, \overset{e}{\boldsymbol{\nu}})$ is 1 and GM is homomorphic with respect to xor ($\oplus$).

VI. Robust Differential Privacy

In this section, we describe how the DCs, mixes, and analyst interoperate to provide robust differential privacy. We explain how each DC tallies the statistic of interest using oblivious counters, and present the details of our HisTorϵ protocol for aggregating these statistics into a differentially private aggregate.

Let Mix$_1$, Mix$_2$, and Mix$_3$ be the three mixes. Mix$_1$ is referred as the master mix and the other two mixes (Mix$_2$ and Mix$_3$) are referred to as the slave mixes. All communication is assumed to be through secure TLS channels. The process of aggregating the individual DC counters is as follows:

Parameter initialization. At the beginning of every epoch, Mix$_1$ generates five cryptographically secure random seeds: the common shuffling seed s, the common random vector seeds p and q, and the pairwise mix seeds x_2 and x_3. It transmits $\langle x_3, p, q, s\rangle$ to Mix$_2$ and $\langle x_2, p, q, s\rangle$ to Mix$_3$. Then, Mix$_2$ generates a cryptographically secure random pairwise mix seed x_1 and transmits $\langle x_1\rangle$ to Mix$_3$.

Query initialization. The analyst formulates a query, and transmits it to the master mix. It specifies the privacy parameter ϵ and, in the case of a histogram query, a set of b bins $\langle [L_1, U_1), \ldots, [L_b, U_b)\rangle$. We discuss the practical aspects of selecting an appropriate value for ϵ in §XI.

Query forwarding. The analyst then transmits the query to the master mix Mix$_1$, which in turn forwards the query to the DCs. The master mix maintains a list, L_c, of the DCs that acknowledged the request.

DC statistics gathering and response. For each query, each data collector D collects statistics during the course of the epoch using three sets of oblivious counters (see §V). There is one set of oblivious counters for each of the three mixes.

At the conclusion of the epoch, the DC performs the following operations:

(i) D maintains a series of encrypted oblivious counters $\overset{e}{\mathbf{v}}_i = \langle E_+(v_{i,1}), \ldots, E_+(v_{i,b})\rangle$ for $1 \le i \le 3$, where each element of $\overset{e}{\mathbf{v}}_i$ is encrypted with Mix$_i$'s GM public key. To ease notation, we refer to the non-encrypted bit vector $v_{i,1}, \ldots, v_{i,b}$ as M.
(ii) D chooses b-bit random binary vectors $R, R_1, R_2, R_3 \in_r \{0,1\}^b$, where $\in_r$ denotes uniformly random selection.
(iii) D computes $R'_i = R \oplus R_i$, $1 \le i \le 3$.

(iv) D encrypts the bits of R with Mix$_i$'s GM public key and multiplies them individually with bits of $\overset{e}{\mathbf{v}}_i$ to obtain $\overset{e}{\mathbf{p}}_i = \langle p_{i,1}, \ldots, p_{i,b} \rangle$ for $1 \leq i \leq 3$.

(v) Finally, D sends the four-tuple of ciphertext $\langle \overset{e}{\mathbf{p}}_1, R'_1, R_2, R_3 \rangle$ to Mix$_1$, $\langle \overset{e}{\mathbf{p}}_2, R_1, R'_2, R_3 \rangle$ to Mix$_2$ and $\langle \overset{e}{\mathbf{p}}_3, R_1, R_2, R'_3 \rangle$ to Mix$_3$.

Note that $\overset{e}{\mathbf{p}}_i$ is the GM encryption of the DC response M xor'ed with R, as GM is a homomorphic encryption scheme. Also, R is computed such that $R = R_1 \oplus R'_1 = R_2 \oplus R'_2 = R_3 \oplus R'_3$. Crucially, each Mix$_i$ receives an encrypted copy of $M \oplus R$, but does not have enough information to unmask (decrypt) M.

In summary, during this phase, each DC xor-encrypts its oblivious counters with a random value, and transmits that ciphertext plus shares of the xor'd random value to each of the mixes.

Mix noise addition and forwarding. Each mix on receiving the four-tuple ciphertext $CT = \langle C_1, C_2, C_3, C_4 \rangle$ from a DC (see step (v) above), checks the legitimacy of C_1. A legitimate GM encrypted value must have its Jacobi symbol equal to '+1', so a mix can easily and efficiently detect malformed responses. If the DC's response is not legitimate, a mix discards it. Otherwise, it decrypts C_1 using its GM private key and obtains the DC response M masked with R. The three mixes then synchronize the list of DCs that have responded. The master mix removes DCs that are not in list L_c. Let the total number of common DCs that have responded be c. To preserve the privacy of the DCs, the mixes collaboratively add n noisy four-tuples, where n is derived using Eq. 1 (where, $\delta = 10^{-6}/c$).

In order to make a noisy tuple and the DC responses indistinguishable, mixes use an efficient xor encryption:

Mix 1:

(i) Chooses random b-bit binary strings P using seed p, Q using seed q, $\mathcal{R}_2$ using pairwise common seed x_2 and $\mathcal{R}_3$ using pairwise common seed x_3.

(ii) Computes $\mathcal{R}'_1 = P \oplus \mathcal{R}_2 \oplus \mathcal{R}_3$.

Mix 2:

(i) Chooses random b-bit binary strings P using seed p, Q using seed q, $\mathcal{R}_1$ using pairwise common seed x_1 and $\mathcal{R}_3$ using pairwise common seed x_3.

(ii) Computes $\mathcal{R}'_2 = P \oplus \mathcal{R}_1 \oplus \mathcal{R}_3$.

Mix 3:

(i) Chooses random b-bit binary strings P using seed p, Q using seed q, $\mathcal{R}_1$ using pairwise common seed x_1 and $\mathcal{R}_2$ using pairwise common seed x_2.

(ii) Computes $\mathcal{R}'_3 = P \oplus \mathcal{R}_1 \oplus \mathcal{R}_2$.

Now, Q is indistinguishable from decrypted C_1, as Q is of the form $M \oplus R$ for some random M and $R = P \oplus \mathcal{R}_1 \oplus \mathcal{R}_2 \oplus \mathcal{R}_3$. Note that, each mix exactly knows only two of the three random vectors $\mathcal{R}_1, \mathcal{R}_2$ and $\mathcal{R}_3$. Therefore, each mix does not know R and hence, the noise that is being added. In other words, mixes add noise, but do not know the values of the noise that they contribute.

Each mix repeats steps (i) and (ii) until all n noisy tuples are generated. Each mix Mix$_i$ then arranges the c DC four-tuples and the n noisy four-tuples, row-wise into four matrices $\langle M_{i,1}, M_{i,2}, M_{i,3}, M_{i,4} \rangle$. Let $M_i = \langle M_{i,1}, M_{i,2}, M_{i,3}, M_{i,4} \rangle$, $1 \leq i \leq 3$. The mixes then shuffle the columns of each matrix in M_i independently, using common shuffling seed s. This shuffling prevents a DC from being identified, and eliminates a potential covert channel. Finally, each Mix$_i$ forwards M_i to the analyst. The master mix in addition forwards the list L_c of DCs that had taken part.

Aggregate calculation. Upon receiving $M_i = \langle M_{i,1}, M_{i,2}, M_{i,3}, M_{i,4} \rangle$, $1 \leq i \leq 3$, the analyst first checks whether the mixes have tampered any DC responses by verifying if:

$$M_{1,1} \overset{?}{=} M_{2,1} \overset{?}{=} M_{3,1} \tag{4}$$

$$M_{2,2} \overset{?}{=} M_{3,2} \tag{5}$$

$$M_{1,3} \overset{?}{=} M_{3,3} \tag{6}$$

$$M_{1,4} \overset{?}{=} M_{2,4} \tag{7}$$

$$M_{1,2} \oplus M_{2,2} \overset{?}{=} M_{2,3} \oplus M_{3,3} \overset{?}{=} M_{3,4} \oplus M_{1,4} \tag{8}$$

If any of the equalities in Eqs. 4 through 8 does not hold, the analyst rejects the response. In such a case, attribution can be performed if exactly one of the mixes is malicious—the mix that does not agree on the equalities with the other two mixes is malicious.

If the equalities hold, then the analyst computes:

$$\bar{M} = M_{1,1} \oplus M_{1,2} \oplus M_{2,2} \tag{9}$$

Finally for every bin j, $1 \leq j \leq b$, the analyst computes the noisy aggregate $\mathbf{a}_j$ as follows:

$$\mathbf{a}_j = \sum_{k=1}^{c+n} \bar{M}[k,j] - n/2 \tag{10}$$

VII. Privacy and Security Analysis

We next argue that HisTorϵ does not reveal any DC statistics to an adversary. We then discuss the privacy guarantees offered by HisTorϵ and evaluate the efficacy of various potential attacks.

A. Security Sketches

We argue the security of the protocol in parts: at the DC, at the mixes and at the analyst. The communication between any two participants (the DCs and the mixes, or the mixes and the analyst) are through TLS channels and are assumed to be secure against eavesdropping. Therefore, we consider the security of the statistics while they are stored on the participants and not while they are in transit on the network.

Claim 1. *Oblivious counters guarantee both confidentiality and integrity of the statistics being collected.*

Proof Sketch. An oblivious counter $\overset{e}{\mathbf{v}}$ is encrypted with a mix's GM public key. By the security of GM encryption, the oblivious counter cannot be decrypted without knowledge of the mix's private key. Moreover, a legitimate GM-encrypted value must have its Jacobi symbol equal to '+1', and hence a counter cannot be malformed by flipping random bits. In other words, GM encryption ensures that the values will only be decrypted as either a `0` or a `1`.

Therefore, the only way the counters can be tampered is by encrypting them to random legitimate GM-encrypted values. This scenario is equivalent to a malicious DC reporting erroneous data. Even in this case, each malicious relay can contribute at most 1 to each bin in its counter. The maximum influence over the aggregate is therefore bounded by the number of malicious DCs. Thus, it follows that the oblivious counters guarantee both confidentiality and integrity of the statistics being collected.

Claim 2. *A mix cannot learn DC statistics or manipulate them without detection.*

Proof Sketch. Each Mix_i knows $\langle M \oplus R \rangle$ from the DCs. The $M \oplus R$ is a one-time pad with secret key R, where R is a random vector that can be obtained from any of the three pairs of random shares: $R_1 \oplus R'_1$, $R_2 \oplus R'_2$ or $R_3 \oplus R'_3$. Here, each Mix_i has exactly one random share from each pair—R'_1, R_2, R_3 in case of Mix_1, R_1, R'_2, R_3 in case of Mix_2 and R_1, R_2, R'_3 in case of Mix_3. Therefore, each mix does not have enough information to unmask M and cannot learn any of the DCs' statistics.

A malicious Mix_i, $1 \leq i \leq 3$ can tamper any of the four-matrices $M_{i,1}, M_{i,2}, M_{i,3}$ or $M_{i,4}$ it receives. We use a tainting technique to prove that the mixes cannot modify any of the matrices without detection. We say that a matrix is tainted with an non-zero impurity if a mix modifies it. Without loss of generality, let us assume that Mix_1 is malicious. Let W, X, Y, and Z be the non-zero impurities used to taint $M_{1,1}, M_{1,2}, M_{1,3}$, and $M_{1,4}$, respectively. All matrices from Mix_2 and Mix_3 are tainted with zero, as only Mix_1 is assumed to be malicious. If suppose, Mix_1 can manipulate the matrices without detection, then all the equalities in Eq. 4 - 8 hold. Therefore, Eq. 4, 6, 7 and 8 are tainted with W, X, Y, and Z as follows:

$$W = 0 \tag{11}$$
$$Y = 0 \tag{12}$$
$$Z = 0 \tag{13}$$
$$X \oplus 0 = 0 = 0 \oplus Z \tag{14}$$

From Equations 11 through 14, we can conclude that $W = X = Y = Z = 0$. This is a direct contradiction to our assumption that W, X, Y, and Z are non-zero impurities. Therefore, a mix cannot manipulate the DC responses without detection.

Claim 3. *An analyst cannot learn any DC statistics.*

Proof Sketch. The analyst can compute $\bar{M}$ from the mixes. $\bar{M}$ contains the DC responses and differentially private noise, and is shuffled column-wise using a cryptographic random seed s that is not known to the analyst. The differentially private noise is indistinguishable from a DC response by the way it is generated: A noise vector Q is of the form $M \oplus R$ for some random M, and $R = P \oplus \mathcal{R}_1 \oplus \mathcal{R}_2 \oplus \mathcal{R}_3$. Also the random vectors $\mathcal{R}'_1, \mathcal{R}'_2$ and $\mathcal{R}'_3$ are generated such that $\mathcal{R}_1 \oplus \mathcal{R}'_1 = \mathcal{R}_2 \oplus \mathcal{R}'_2 = \mathcal{R}_3 \oplus \mathcal{R}'_3 = R$. Therefore, the analyst sees a pseudorandom permutation of the DC responses and differentially private noise in the columns of $\bar{M}$. By the security of pseudorandom permutation, the columns of $\bar{M}$ cannot be distinguished from a random permutation with practical effort. Therefore, the analyst cannot learn any DC response from $\bar{M}$.

Claim 4. *A mix cannot learn the actual aggregate by subtracting the noise from the published aggregate.*

Proof Sketch. A mix adds a noise vector Q of the form $M \oplus R$ for some random M, and $R = P \oplus \mathcal{R}_1 \oplus \mathcal{R}_2 \oplus \mathcal{R}_3$. A mix exactly knows only two of the three random vectors $\mathcal{R}_1, \mathcal{R}_2$, and $\mathcal{R}_3$. So, a mix does not know R and hence the noise that is being added. Therefore, a mix cannot learn the actual aggregate by subtracting the noise from the published aggregate.

B. Privacy Analysis

Data collector. The DCs can be malicious and report "junk" statistics. The maximum absolute distortion in the final aggregate result is bounded by the number of malicious DCs. This bound is much more lenient than PrivEx: the adversary needs to compromise many DCs to substantially distort the result provided by HisTorϵ; in PrivEx, a single malicious data collector can significantly influence the aggregate result (see §III).

When two or more DCs collude, they learn no more information by pooling their oblivious counters. This follows from the guarantees offered by the differential privacy mechanism [6]. HisTorϵ protects a single honest DC's statistics even when all other DCs are malicious. Also, the statistics collected at the honest DCs are preserved from the actions of a misbehaving DCs as long as the security of the GM encryption scheme remains intact.

Even when a DC colludes with a mix or an analyst, it learns no more information than what is already known to it.

Mixes. The mixes can be malicious and can report "junk" statistics or refuse to add noise. However, as long as at least two of the three mixes are honest, we can identify the malicious mix. Moreover, the malicious mix cannot learn any DC statistics as shown in §VII-A.

Even when two non-colluding mixes are malicious, we can still detect such an attack, but cannot attribute it. Also, in either of these cases, the mixes do not know the noise added and cannot learn the actual aggregate.

However, when two or more malicious mixes collude, they can trivially learn all DC responses without detection. We discuss this threat in more detail in §XI.

Analyst. The analyst cannot learn any DC's response as shown in §VII-A. However, a malicious mix can share the shuffling seed s with the analyst. This allows the analyst to learn all the DC responses.

C. Attack Resilience

To complete our security analysis, we consider three types of attacks against HisTorϵ: compulsion attacks, denial-of-service (DoS) attacks, and correlation attacks.

Compulsion attacks. A DC can be compelled to reveal its counters through a legal order or extra-legal compulsion. HisTorϵ mitigates this threat by encrypting the counters with the mixes' public key. A DC cannot decrypt its own oblivious counters.

The adversary can also compel the mixes to decrypt the DCs' statistics. However, each mix receives the client response masked with the random vector R, and does not have enough information to unmask it (each mix has either R_i or R'_i for $1 \leq i \leq 3$, but not both R_i and R'_i).

To successfully conduct a compulsion attack, the adversary would have to first compel the DC to release its oblivious counters, and then further compel a mix to perform the decryption. While such an attack is technically feasible, we imagine that compelling a mix to perform a decryption would garner significant attention; as a rough equivalent, this is analogous to compelling a Tor directory authority to release its signing keys.

The adversary may compel the analyst to release statistics before the analyst aggregates the result. However, this attack is fruitless since the analyst has only the noised data, with differentially private guarantees.

DoS attacks. A DC can refuse to participate in a query or submit a malformed vector. This is easily mitigated by discarding the particular DC's response. Consequently, malicious DCs cannot cause denial-of-service in a round of HisTorϵ.

Mixes may also DoS HisTorϵ queries. The availability of the aggregate is guaranteed as long as at least two of the three mixes participate.

Correlation attacks. An attacker can combine the collected statistics with some auxiliary information such as observations of a target user's network traffic and perform a correlation attack. However, the differential privacy mechanism [6] provides strong privacy guarantees against such a threat.

VIII. A Practical Consideration: Guided Binning

When posing a histogram query, an analyst needs to define the bin widths. Determining useful bin widths for a histogram is highly subjective, and can be difficult if the analyst does not have an intuition as to the overall shape of the histogram.

Here, as a more practical contribution, we present a simple algorithm for partially automating this process. Conceptually, the algorithm operates by modifying the definition of bin widths (that is, by splitting and joining bins) in a sequence of epochs until a useful histogram is obtained. Since each epoch lasts one hour, our goal is to quickly converge on a useful bin width definition.

The analyst runs the guided binning algorithm until it gets a "satisfactory" histogram of the noised results. In our experiments, we find that we achieve a useful histogram after three or four epochs. Importantly, once a useful definition of bin widths is achieved for a given query—e.g., the number of connections seen by guard relays—this definition tends to hold over time. That is, the guided binning algorithm is most useful when issuing a new type of query or when the results of a query indicate unexpected results.

```
// $w$ - New Bin Width List
// $cur\_w$ - Current Bin Width
// $max$ - UB of Last Bin in Previous Iteration
proc $gcdBinWidth(w,\ cur\_w,\ max)$
if $!w$ then
    $g \leftarrow cur\_w$ // List Empty
else
    $minw \leftarrow \min(w)$
    $g \leftarrow gcd(minw,\ cur\_w)$
    if $g = 1$ or $g < \min(minw,\ cur\_w)$ then
        $i \leftarrow 1$
        $sqrt \leftarrow \sqrt{minw} + 1$
        /* compute greatest factor of $minw$ lesser than or equal to $cur\_w$ */
        while $i \leq sqrt$ do
            if $minw \% i = 0$ and $minw / i \leq cur\_w$ then
                $g \leftarrow minw/i$
                break
            end
            $i \leftarrow i + 1$
        end
        // g is 1 or auxiliary bins > 15000
        if $i = sqrt + 1$ or $max/g > 15000$ then
            $g \leftarrow minw$
        end
    end
end
return $g$
```

Procedure GCDBinwidth

```
// $g$ - GCD returned by gcdBinwidth()
// $cur\_w$ - Current Bin Width
proc $adjustBinWidth(g,\ cur\_w)$
if $cur\_w < g$ then
    $cur\_w \leftarrow g$
else
    // $cur\_w$ not a multiple of $g$
    if $cur\_w \% g \neq 0$ then
        // make $cur\_w$ a multiple of $g$
        if $(cur\_w \% g) < (g - (cur\_w \% g))$ then
            $cur\_w \leftarrow cur\_w - (cur\_w \% g)$
        else
            $cur\_w \leftarrow cur\_w + (g - (cur\_w \% g))$
        end
    end
end
return $cur\_w$
```

Procedure AdjustBinwidth

We refer to each run of the guided binning algorithm as an iteration.

The first iteration takes two parameters as input: b, the number of bins; and e, the total estimate of the statistics (e.g., total bandwidth, total number of client connections, etc.) being collected. Equal width bins are assigned for this first iteration—the lower bound of first bin is set to 0, a bin-width of $\lfloor e/b \rfloor$ is used for the first $b-1$ bins and the upper bound of last bin is set to e.

If more iterations are needed, the next iteration uses the bin width distribution and the noised result of the previous iteration to obtain a more optimal bin width distribution. The guided binning algorithm first computes the mean value, k of the noised distribution. Then, in the lexical ordering of bins, starting from bin 1, it splits all bins that have a value r_i greater than k into $\lfloor r_i/k \rfloor$ bins. The algorithm also merges all consecutive bins that have a value less than k until their combined value does not exceed k.

When a bin is split or when bins are merged, the algorithm executes the GCDBinwidth procedure. The GCDBinwidth procedure chooses an optimal GCD value g (of the bin widths up to that point) such that the total number of auxiliary bins (see §V-B) is less than 15000. Then the guided binning algorithm executes the AdjustBinwidth procedure that makes the current new bin width a multiple of the optimal GCD g.

The algorithm can be terminated during any iteration in which the analyst obtains a "satisfactory" histogram of the noised results.

IX. Implementation and Evaluation

We constructed an implementation of HisTorϵ in Python to verify our protocols' correctness, assess the utility of the noised aggregates, and measure the system's overheads. We implement GM encryption using the Python libnum library[6] with a modulus size of 1024 bits. Our PRF is based on AES in CFB mode, supported by the PyCrypto cryptographic library. Mixes perform shuffle operations by applying the Fisher-Yates algorithm, using the AES-based PRF as its source of randomness.

As a system-wide parameter, we set $\epsilon = 1$ for all experiments unless otherwise indicated. We note that this offers more privacy than the experimental setting ($\epsilon = 5$) used by Chen et al. [6]. For histogram queries, we semi-automate the process of selecting appropriate bin widths using the human-guided bin splitting algorithm described in §VIII.

Experiments were carried out on a 16-core AMD Opteron machine with 32GB of RAM running Linux kernel 3.10.0. Our implementation of HisTorϵ is currently single-threaded. Although HisTorϵ's computational costs are insignificant (see §IX-D), certain operations are embarrassingly parallel—in particular, encrypting and decrypting the elements of the $\overset{e}{\mathbf{v}}$ vectors—and could likely further benefit from parallelization.

We instantiated three HisTorϵ mix instances, one HisTorϵ analyst instance, and all DCs on our 16-core server. In a real deployment, these instances would all be distributed. Google Protocol Buffers [29] were used for serializing messages, which were communicated over TLS connections between HisTorϵ components. We use Python's default TLS socket wrapper for securing communication. All communication took place over the local loopback mechanism (localhost).

Our experiments do not run actual Tor relay or client code. As explained in more detail next, we derive our unnoised statistics from existing published data sets[7]. This data is used as input to our DC instances, which then run HisTorϵ protocols to enable the analyst to obtain (noised) aggregate results.

[6]https://github.com/hellman/libnum

A. *Queries and Data Sets*

We evaluate HisTorϵ by considering three histogram queries: the number of client connections as observed by Tor guards, the amount of bandwidth used by Tor guards, and the amount of exit bandwidth used by Tor exit relays. As our ground truth, we use data from both the Tor Compass and the Tor Metrics Portal.

Number of client connections. For the number of client connections, each Tor guard acts as a DC. In total, we instantiate 1839 DCs—the total number of guards reported by the Tor Compass with a non-zero selection probability.

We derive our "ground truth" by considering the total number of direct users (T) connecting to Tor as reported by the Tor Metrics Portal over the period of July 26th through July 30th, 2016. We assign $p_i \cdot T$ client connections to each guard relay i, where p_i is the guard selection probability for guard i as reported by the Tor Compass.

Bandwidth used by guards/exits. Similarly, as our ground truth for the bandwidth observed by guards (resp. exits), we consider the total guard (resp. exit) bandwidth (B) reported by the Tor Metrics Portal over the same five-day time period. Each guard (resp. exit) acts as a DC, and is assigned a bandwidth cost of $(p_i \cdot B)$, where p_i is the selection probability of the guard (resp. exit). We instantiate 1839 DCs when measuring guard bandwidth, and 924 DCs in the case of exit bandwidth. The latter is the number of exits reported by the Tor Compass that have a non-zero selection probability.

We do not argue that the above procedures yield perfect ground truth. We apply them to derive a gross approximation of the distributions of client connections and bandwidths which can then be used to test the efficacy of HisTorϵ under near-real-world conditions. When deployed, HisTorϵ allows for much more accurate and fine-grained statistics reporting than offered by the Tor Metrics Portal.

B. *Accuracy*

Figure 5 shows the returned histograms for the three queries when applied to the Tor datasets. The Figure plots the results of the histogram query after three iterations of the guided binning algorithm (see §VIII). Other iterations exhibited similar accuracy (as measured by the difference between the noised and unnoised distributions), but had arguably less useful bin definitions.

HisTorϵ reports the "Noised" values shown in Figure 5. As is clear from the Figure, these noised values closely resemble

[7]**Note on ethical considerations:** We evaluate HisTorϵ using only already published data from the Tor Compass and Tor Metrics Portal. This data, by its nature, is derived from the activities of human subjects. Because the data has already been published and contains no personally-identifiable information (PII), IRB approval was not required by our institution(s). Much more importantly, the data itself is carefully collected by the Tor Project in a manner that protects the privacy and anonymity of Tor's users. The Tor Compass and Tor Metrics Portal publish only aggregate information (e.g., at the granularity of a country) that is unlikely to expose information about any particular user. This paper advocates for even stronger privacy protections.

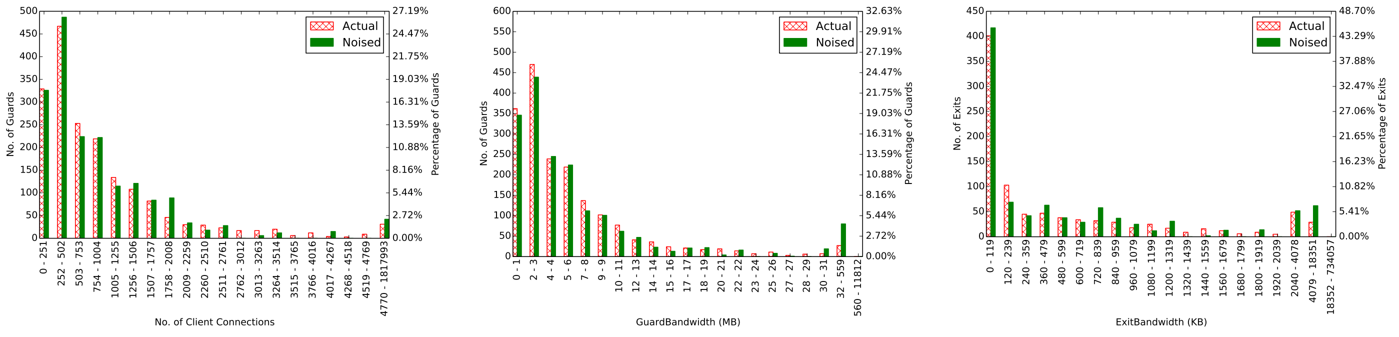
Fig. 5. The aggregate histogram results returned by HisTorϵ when the analyst issues a query for the number of client connections as observed by guards (*left*), the amount of bandwidth used by guards (*center*), and the amount of bandwidth used by exit relays (*right*). Also shown is the unnoised distribution ("Actual").

TABLE I. DISTANCES BETWEEN THE "ACTUAL" AND "NOISE" DISTRIBUTIONS SHOWN IN FIGURE 5.

Distance function	No. Client Conn	GuardBW	ExitBW
R^2	0.98466	0.98290	0.96970
Bhattacharyya	0.01820	0.01179	0.02542

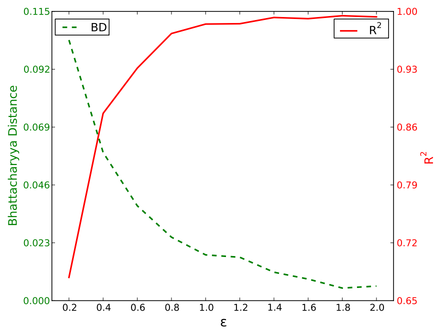
Fig. 6. The distance between the actual and noised distributions, as measured by the Bhattacharyya distance (left y-axis) and R^2 (right y-axis).

those of the unnoised ("Actual") ground truth data. Looking at just the "Noised" values, an analyst can clearly obtain useful information about the distributions of client connections, guard bandwidths, and exit bandwidths.

As a more quantifiable indicator of the *closeness* between the noised and unnoised distributions, we consider both the coefficient of determination (also called the R^2 distance) and the Bhattacharyya Distance [3]. The latter measures the divergence between two probability distributions and ranges from 0 (identical distributions) to ∞. An R^2 value of 1 indicates perfect prediction. When no correlation exists, $R^2 = 0$. Table I reports the distances between the actual and noised distributions. Our results highlight that even with a conservative setting of $\epsilon = 1$, HisTorϵ produces highly accurate aggregates.

The tradeoff between accuracy and security is governed by the choice of ϵ. We explore this space by varying ϵ between 0.2 and 2.0 for the connection count query. As shown in Figure 6, we find that varying ϵ has little effect on accuracy. The overall variation in R^2 (resp. Bhattacharyya) distance between $\epsilon = 0.2$ and $\epsilon = 2.0$ was only 0.315 (resp. 0.098).

C. Bandwidth Overhead

HisTorϵ incurs communication overhead by transmitting encrypted counters between the DCs and the mixes, and

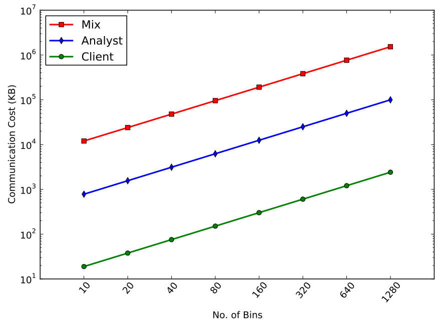
Fig. 7. HisTorϵ's communication cost per epoch (y-axis) as a function of the number of bins (b; shown on the x-axis). Both axes are plotted in log scale.

encrypted matrices between the mixes and the analyst. To be practical, a statistics gathering system should impose a low communication overhead for the DCs, since relays are already a bandwidth-limited resource in Tor [8]. We envision that mixes and the analyst are dedicated resources for HisTorϵ, and our goal is to not incur unreasonable bandwidth requirements for these components.

We explore HisTorϵ's bandwidth costs by varying the number of bins b in a query. The values of the bins for the type of query (class vs. histogram) do not affect the communication cost, as the DCs only transmit $\overset{e}{\mathbf{v}}$ (and not $\overset{e}{\boldsymbol{\nu}}$) for both query types. In our bandwidth measurements, we fix the number of DC relays at 1839.

Figure 7 shows the average communication costs for a DC, mix, and analyst. For up to 80 bins, the communication cost for each DC is fairly modest and is approximately 150 KB per hour (or about 42 Bps). Generally, we anticipate the number of bins to be around 20, although this can vary depending upon the analyst's query. As a potential point of interest, the histograms shown in Figure 5 were derived using the guided binning process, and resulted in 20, 21, and 20 bins (from left to right).

Even when the number of bins is quite large (1280), a DC's communication cost is only 2.4 MB over the course of the hourlong epoch—or 0.67 KBps.

The communication costs are greater for the mixes and the analyst. With 40 bins, each mix consumes 47.8 MB of bandwidth per hour, while the analyst uses 3.1 MB. In our

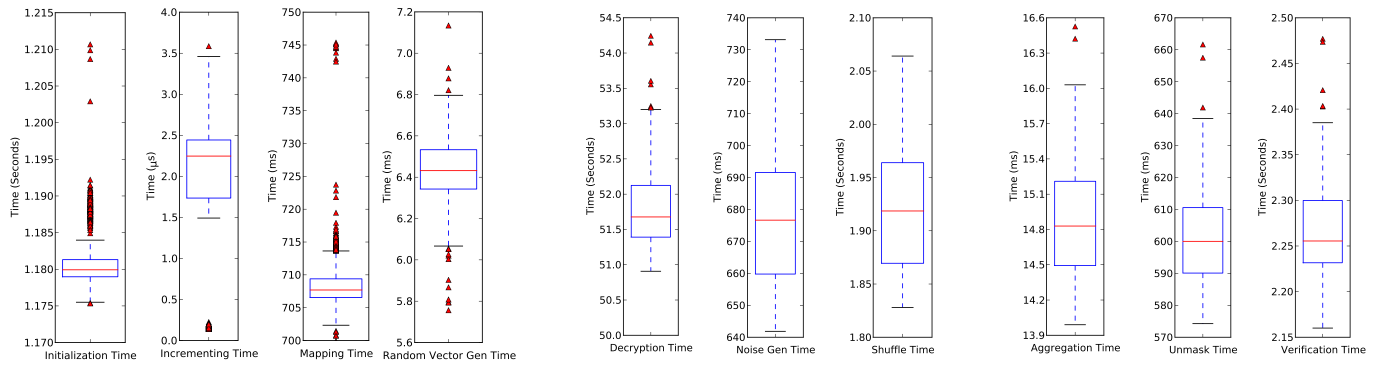

Fig. 8. Microbenchmark results for the DSes (left), mixes (center), and analyst (right). Boxes denote the lower and upper quartile values, with a line at the median. The "whiskers" show the range of the data within 1.5xIQR from the lower and upper quartiles. Outliers are indicated with triangles.

largest binning scenario with 1280 bins, each mix consumes 1.5 GB of bandwidth each hour, or 424.7 KBps; the analyst requires 27.7 KBps.

In summary, we find that for modest number of bins (20-40), HisTorϵ incurs very little bandwidth overhead, and can thus support multiple concurrent queries.

D. Computation Overheads

To understand the computation costs of HisTorϵ, we perform a series of microbenchmarks. In our measurements, we consider the connection counting histogram query with twenty bins and 1839 DCs.

The left side of Figure 8 shows the distribution of the processing overheads for the DCs. The operations involved in maintaining the oblivious counters are initialization, incrementing, and mapping (from $\overset{e}{\boldsymbol{\nu}}$ to $\overset{e}{\mathbf{v}}$). The total costs of these operations is, in the worst case, less than two seconds per hour (on a single core). Additionally, the DCs generate the random vectors $\mathbf{R_i}$ and $\mathbf{R_i'}$ (such that $\mathbf{R} = \mathbf{R_i} \oplus \mathbf{R_i'}$), incurring a worst case processing overhead of approximately 7.2 ms per hour.

The performance overheads for a mix are shown in the center of Figure 8. To explore the range in overheads, we repeat our query 100 times and plot the range of processing costs incurred by the mix over all runs. Here, the operations consist of GM decryption, the generation of noise records, and shuffling the matrices. Each hour, in the worst case, a mix spends approximately one minute of computation for a single query. Using a single core, a mix could thus support at worst 60 simultaneous queries per hour.

The analyst's processing times are shown in the right side of Figure 8. As with the mix, we show the results over 100 iterations of the query. The most burdensome operation is verification of the three matrices. This consumes less than three seconds of processing, in the worst case, per hour.

Overall, the processing costs of operating HisTorϵ is negligible for the relays (DCs) and analyst, and manageable for the mixes.

X. Related Work

Loesing et al. [22] motivate the need for performing statistical analysis of the Tor network. The authors promote privacy-preserving statistics gathering as a means to identify trends in the network, to detect performance bottlenecks, to discover attacks against the network, and to better understand how Tor is being blocked and censored in various regions of the Internet [22].

We are not the first to propose using differential privacy to collect sensitive statistics on Tor. Elahi et al. [15] introduce two variants—one based on secret sharing and another that uses distributed decryption—to privately collect traffic statistics for Tor. PrivEx provides strong, measurable privacy protection even against compromised participants. However, PrivEx does not provide integrity guarantees, and as we demonstrate in §III, even a single malicious DC can cause inaccurate results. In comparison, HisTorϵ also applies differential privacy techniques to privately collect data, but in addition provides strong integrity guarantees about the influence of malicious DCs.

PrivCount [18] extends the PrivEx secret-sharing variant to make it suitable for a small-scale research deployment. It allows multi-phase iterative measurements and expands the privacy notion of PrivEx to simultaneously handle multiple and diverse Tor statistics with an optimal allocation of the ϵ privacy budget. However, even in PrivCount, there are no protections for the integrity of the statistics (see §III).

We use a slight modification of the (ϵ, δ)-differential privacy scheme of Chen et al. [6]. They use a single mix, which they call a *proxy*. This allows a malicious proxy to undetectably alter the data and cause the analyst to reach an inaccurate aggregate result. We detect such manipulation in HisTorϵ by adding redundancy across three mixes. As we prove in §VII, HisTorϵ detects such tampering if at least one of the mixes is honest, and can attribute the misbehavior if two of the three mixes are honest. Additionally, the scheme of Chen et al. is vulnerable to compulsion attacks. HisTorϵ uses oblivious counters to mitigate such risks.

HisTorϵ is partially inspired by SplitX, which executes differentially private queries over distributed data [7]. Like HisTorϵ, SplitX uses the (ϵ, δ)-differential privacy scheme of Chen et al. and uses xor-based encryption to distribute secret shares to separate mixes. In SplitX, both the mixes and the aggregator are assumed to be honest-but-curious. In HisTorϵ, we are able to tolerate a malicious mix by redundantly encoding information in secret shares.

McSherry and Mahajan [25] apply differential privacy techniques to the PINQ programming interface [26] in order to support privacy-preserving network trace analysis. They show that performing trace analysis in a differentially private setting is technically feasible, but do not offer a distributed solution. Combining their network trace analysis techniques with HisTorϵ could potentially allow network operators to identify patterns of misbehavior in Tor. We leave synthesizing these techniques as an exciting avenue of future research.

XI. Discussion and Limitations

In this section, we discuss practical aspects of deploying HisTorϵ, as well as some of the system's limitations.

Detectability, attribution and suitability of the threat model. A malicious mix may attempt to manipulate the results of a query by modifying the inputs it receives from the DCs. As we discuss in §VI and §VII, an analyst can *detect* that misbehavior occurred if at least one of the mixes is honest. Since data is replicated across all three mixes, we can additionally *attribute* the misbehavior to a specific malicious mix if exactly two of the three mixes are honest—the malicious mix will be revealed through its non-conforming output.

The difficulty with performing attribution is that the cases of one malicious mix vs. two malicious mixes can be indistinguishable if, in the latter case, the two mixes perform identical manipulations. Unless it is readily apparent through some other mechanism which mix(es) has been compromised, a reasonable solution once misbehavior is detected is to re-evaluate the security of all three mixes.

More generally, mixes should be carefully selected, since collusion between two or more dishonest mixes compromises data privacy. This is, to some degree, similar to Tor's existing quasi-centralized notion of trust: if a majority of the Tor directory authorities are compromised, then Tor offers no anonymity protections since the directories could advertise only the existence of malicious relays.

Like directory authorities, mixes must therefore be chosen carefully. We envision that the maintainers of the Tor Project could selectively grant permission to operate HisTorϵ mixes to parties. Or, to keep the existing level of trust, the directory authorities could additionally operate mixes.

From the perspective of integrity, it may at first blush seem that HisTorϵ and PrivEx offer similar guarantees—that is, certain nodes must behave honestly to ensure integrity of the query results. We argue that the integrity guarantees offered by HisTorϵ are significantly stronger, since in PrivEx, a single relay can significantly perturb the results of a query. Successfully compromising PrivEx statistics gathering is thus a fairly simple operation since there are no barriers to operating a relay. In contrast, HisTorϵ removes the necessity to trust the data collectors, and instead relies on a much smaller set of nodes (i.e., mixes). Additionally, so long as a single mix is honest, query tampering can be trivially detected.

Selection of ϵ. A consistent problem in schemes that apply differential privacy is selecting an appropriate value of ϵ. In our experimentation, we apply the same conservative value as existing work [25] and set $\epsilon = 1$.

Since ϵ is a relative and not an absolute measure of privacy, an interesting area of future work is to derive *per-query* values of ϵ that are guaranteed to protect individuals with some fixed probability. Lee and Clifton [21] provide one such construction for ϵ-differential privacy. Incorporating this selection process into HisTorϵ (which provides (ϵ, δ)-differential privacy) is an exciting potential future research direction.

In the current implementation of HisTorϵ, the analyst communicates its choice of ϵ to the mixes. Since the number of noise records is proportional to ϵ^{-2}, large values of ϵ offer little security while too small values of ϵ have little benefit to privacy while incurring potentially enormous communication costs. To provide a simple sanity-check, a real-world deployment of HisTorϵ could establish system-wide parameters ϵ_{max} and ϵ_{min} that bound the analyst's choice.

Privacy budget. (ϵ, δ)-differential privacy schemes impose a *privacy budget* whose balance is decremented as a function of δ and ϵ for each issued query. This budget is defined in terms of a static database $\mathcal{D}$ over which queries are issued. In HisTorϵ, counters are zeroed after each epoch, effectively resulting in a new database. This thus significantly reduces the risk of exceeding the privacy budget.

Unfortunately, for certain query types, there may be dependencies in the statistic of interest between epochs that violates this assumption of independence. This further motivates a careful selection of ϵ to minimize this privacy risk.

XII. Conclusion

This paper presents HisTorϵ, a distributed statistics collection system for Tor. Unlike existing work, HisTorϵ provides strong integrity guarantees for collecting data on Tor. In particular, we demonstrate that the influence of a colluding group of malicious data collectors is tightly bounded by the fraction of nodes that they control in the network. More practically speaking, HisTorϵ ensures that a small colluding group of malicious data collectors has negligible impact on the results of statistics queries.

In addition to ensuring integrity, HisTorϵ provides strong privacy guarantees as long as malicious mixes do not collude with a dishonest analyst. HisTorϵ also achieves resistance to compulsion attacks through use of novel oblivious counters.

We demonstrate using real-world data sets and realistic query workloads that HisTorϵ enables highly accurate statistics aggregation, with small bandwidth and computational overheads. Our performance experiments and microbenchmarks indicate that dozens of simultaneous HisTorϵ queries could be supported on a single CPU core. To encourage its use by privacy researchers, we are planning an open-source release of HisTorϵ in the near future.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. We also thank Aaron Johnson, Henry Tan, and Sridhar Venkatesan for the valuable discussions. This paper is partially funded from National Science Foundation (NSF) grants CNS-1149832 and CNS-1527401. The findings and opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] Akamai's State of the Internet Q2 2015 Report, 2015. Available at https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/2015-q2-cloud-security-report.pdf.

[2] K. Bauer, M. Sherr, D. McCoy, and D. Grunwald. ExperimenTor: A Testbed for Safe and Realistic Tor Experimentation. In *USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, August 2011.

[3] A. Bhattachayya. On a Measure of Divergence between two Statistical Population Defined by their Population Distributions. *Bulletin Calcutta Mathematical Society*, 35:99–109, 1943.

[4] A. Biryukov, I. Pustogarov, F. Thill, and R.-P. Weinmann. Content and Popularity Analysis of Tor Hidden Services. In *International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2014.

[5] T.-H. H. Chan, E. Shi, and D. Song. Private and Continual Release of Statistics. *ACM Transactions on Information and System Security (TISSEC)*, 14(3), 2011.

[6] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke. Towards Statistical Queries over Distributed Private User Data. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.

[7] R. Chen, I. E. Akkus, and P. Francis. SplitX: High-performance Private Analytics. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2013.

[8] R. Dingledine and S. Murdoch. Performance Improvements on Tor, or, Why Tor is Slow and What We're Going to Do About It. https://svn.torproject.org/svn/projects/roadmaps/2009-03-11-performance.pdf, March 2009.

[9] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium (USENIX)*, August 2004.

[10] C. Dwork. Differential Privacy. *Automata, Languages and Programming*, pages 1–12, 2006.

[11] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our Data, Ourselves: Privacy via Distributed Noise Generation. In *Advances in Cryptology (Eurocrypt)*, 2006.

[12] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential Privacy under Continual Observation. In *ACM Symposium on Theory of Computing (STOC)*, 2010.

[13] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[14] T. Elahi, K. Bauer, M. AlSabah, R. Dingledine, and I. Goldberg. Changing of the Guards: A Framework for Understanding and Improving Entry Guard Selection in Tor. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2012.

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

[16] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[17] R. Jansen and N. Hopper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

[18] R. Jansen and A. Johnson. Safely Measuring Tor. In *ACM Conference on Computer and Communications Security (CCS)*, 2016.

[19] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. Users Get Routed: Traffic Correlation on Tor By Realistic Adversaries. In *ACM Conference on Computer and Communications Security (CCS)*, November 2013.

[20] D. Kedogan, D. Agrawal, and S. Penz. Limits of Anonymity in Open Environments. In *Information Hiding Workshop (IH)*, 2002.

[21] J. Lee and C. Clifton. How Much is Enough? Choosing ε for Differential Privacy. In *International Conference on Information Security*, 2011.

[22] K. Loesing, S. J. Murdoch, and R. Dingledine. A Case Study on Measuring Statistical Data in the Tor Anonymity Network. In *Financial Cryptography and Data Security (FC)*. 2010.

[23] N. Mathewson. Some Thoughts on Hidden Services (Tor Blog Post), 2014. Available at https://blog.torproject.org/blog/some-thoughts-hidden-services.

[24] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker. Shining Light in Dark Places: Understanding the Tor Network. In *Privacy Enhancing Technologies Symposium (PETS)*, 2008.

[25] F. McSherry and R. Mahajan. Differentially-private Network Trace Analysis. *ACM SIGCOMM Computer Communication Review*, 41(4):123–134, 2011.

[26] F. D. McSherry. Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2009.

[27] M. Perry. The Trouble with CloudFlare (Tor Blog Post), March 2016. Available at https://blog.torproject.org/blog/trouble-cloudflare.

[28] M. Prince. The Trouble with Tor (CloudFlare Blog Post), March 2016. Available at https://blog.cloudflare.com/the-trouble-with-tor/.

[29] Protocol Buffers. https://developers.google.com/protocol-buffers/.

[30] C. Soghoian. Enforced Community Standards For Research on Users of the Tor Anonymity Network. In *Workshop on Ethics in Computer Security Research (WECSR)*, 2011.

[31] Tor Project, Inc. Tor Metrics Portal. https://metrics.torproject.org/.

[32] Tor Research Safety Board. Available at https://research.torproject.org/safetyboard.html.

[33] C. Wacek, H. Tan, K. Bauer, and M. Sherr. An Empirical Evaluation of Relay Selection in Tor. In *Network and Distributed System Security Symposium (NDSS)*, February 2013.

[34] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, and E. Weippl. Spoiled Onions: Exposing Malicious Tor Exit Relays. In *Privacy Enhancing Technologies Symposium (PETS)*, 2014.