Stormy: Statistics in Tor by Measuring Securely

Ryan Wails
ryan.wails@nrl.navy.mil
U.S. Naval Research Laboratory

Aaron Johnson
aaron.m.johnson@nrl.navy.mil
U.S. Naval Research Laboratory

Daniel Starin
dstarin@perspectalabs.com
Perspecta Labs

Arkady Yerukhimovich*
arkady@gwu.edu
George Washington University

S. Dov Gordon
gordon@gmu.edu
George Mason University

ABSTRACT

Tor is a tool for Internet privacy with millions of daily users. The Tor system benefits in many ways from information gathered about the operation of its network. Measurements guide operators in diagnosing problems, direct the efforts of developers, educate users about the level of privacy they obtain, and inform policymakers about Tor's impact. However, data collection and reporting can degrade user privacy, contradicting Tor's goals. Existing approaches to measuring Tor have limited capabilities and security weaknesses.

We present Stormy, a general-purpose, privacy-preserving measurement system that overcomes these limitations. Stormy uses secure multiparty computation (MPC) to compute *any* function of the observations made by Tor relays, while keeping those observations secret. Stormy makes use of existing efficient MPC protocols that are secure in the malicious model, and in addition it includes a novel *input-sharing protocol* that is secure, efficient, and fault tolerant. The protocol is non-interactive, which is consistent with how relays currently submit measurements, and it allows the relays to go offline after input submission, even while ensuring that an honest relay will not have its input excluded or modified. The input-sharing protocol is compatible with MPC protocols computing on authenticated values and may be of independent interest.

We show how Stormy can be deployed in two realistic models: (1) run primarily by a small set of dedicated authorities, or (2) run decentralized across the relays in the Tor network. Stormy scales efficiently to Tor's thousands of relays, tolerates network churn, and provides security depending only on either Tor's existing trust assumption that at least one authority is honest (in the first model) or the existing assumption that a large fraction of relay bandwidth is honest (in the second model).

We demonstrate how to use the system to compute two broadly-applicable statistics: the median of relay inputs and the cardinality of set-union across relays. We implement Stormy and experimentally evaluate system performance. When Stormy is run among authorities we can perform 151 median computations or 533 set-union cardinalities over 7,000 relay inputs in a single day. When run among the relays themselves, Stormy can perform 36 median computations or 134 set union cardinalities per day. Thus, both deployments enable non-trivial analytics to be securely computed in the Tor network.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**.

KEYWORDS

Tor; secure multi-party computation; cryptographic protocols

ACM Reference Format:
Ryan Wails, Aaron Johnson, Daniel Starin, Arkady Yerukhimovich, and S. Dov Gordon. 2019. Stormy: Statistics in Tor by Measuring Securely. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3345650

1 INTRODUCTION

The Tor network [26] is perhaps the most popular tool for private and open communication on the Internet. As of 2018-03-31, Tor has an estimated two million daily users from around the world, and its almost 7,000 relays forward over 100 Gbps of traffic [5]. Tor also protects the privacy and integrity of over 60,000 onion services, which benefit from Tor's anonymity, end-to-end encryption, and secure name lookup. Statistics such as these provide some insight into how Tor is being used and how well it is performing, which guides software developers in improving Tor, informs policymakers about Tor's social impact, and helps users understand who else is using Tor and thus what kind of anonymity it provides. However, gathering such statistics conflicts to some extent with Tor's goal of providing privacy to its users. As a result, Tor collects relatively little data about itself, and it protects what it does collect by aggregating it and limiting its accuracy. This decision has left Tor unable to quickly determine when it is under attack [13, 44], how its traffic is being blocked or degraded [50, 69], and for what purposes Tor is being used [8, 12, 42, 54].

Several recent tools have been developed to allow Tor to gather network statistics while maintaining individual user privacy [30, 31, 42, 56]. These tools apply secure aggregation and differential privacy to produce statistics in a privacy-preserving way. The functionality of these tools is limited, however, which makes them unsuitable for many useful classes of measurements, such as statistics robust to outliers and non-linear data-sketching techniques.

*Part of this work was done while the author was at MIT Lincoln Laboratory.

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3345650

The technical report for this work containing all appendices and proofs is available online.

We present a system that provides fully-general distributed measurement and monitoring on Tor by making use of secure multi-party computation (MPC). This system provides Tor with tools to intentionally choose the level of network transparency that is most consistent with its goal of providing online privacy and freedom.

To make sure that our system is compatible with Tor, we aim to rely only on security assumptions that are inherent in the secure use of Tor. Specifically, we consider two different deployment models each relying on a different, standard trust assumption in the Tor network. The first deployment model we consider, the Authority Model, considers running the MPC protocol among a small number of dedicated authorities. In this model, we only assume that at least one of the authorities is honest, a standard assumption for directory authorities in Tor. This model allows us to show the performance that can be achieved under a strong, but standard trust assumption for Tor. The second deployment model, the Relay Model, looks to relax this trust assumption to the minimal requirement that a reasonable fraction (e.g., 75%) of Tor's total bandwidth capacity is controlled by honest relays. In this deployment model we execute our MPC protocols directly over nearly 7,000 relays while aiming to maximize the throughput of the MPC.

This second deployment model required the development of new MPC protocols, specialized to large numbers of parties, and relying on an honest fraction of bandwidth, as opposed to the usual assumption of an honest number of parties. While the field of MPC has seen tremendous progress in the last decade, this work marks the first attempt to tackle the practical challenges that arise in a secure computation involving thousands of parties. While there is a long line of work studying how to handle larger computations from a *theoretical* perspective [20, 25, 35, 36, 63, 72], mainly relying on various committee-election procedures, this work does not consider several realistic concerns about node churn, bandwidth, and memory that we address in our work. Moreover, the largest MPC experiment performed to date [68] involves 128 parties. However, the Tor network at the end of March 2018 consisted of almost 7,000 relays, and that number continues to grow.

One major practical challenge we address is the uneven distribution of bandwidth. When dealing with 7,000 independent parties in a globally-distributed system, inevitably, not all parties will have equal resources. To deal with this issue, we show a simple way to elect committees of parties that ensures low communication cost while also guaranteeing high bandwidth utilization, even when participants have highly varying bandwidth allocations (Section 4.1). Specifically, we rely on a fairly simple observation. Many MPC protocols can be divided into offline and online phases [10, 15, 22, 23, 49, 68]. The offline phase requires quadratic communication, but it is data independent, and can be performed in parallel. So, instead of using a single committee, as done by prior work, we assign each party to multiple committees proportional to the amount of bandwidth that they have, ensuring that parties with higher bandwidth are not limited by the reduced bandwidth of smaller parties.

While the above techniques should find application in other large-scale MPC implementations, assigning parties to multiple committees also provides an important security benefit in the context of Tor. Tor's trust assumptions are different than those typically made for MPC protocols: its security fundamentally requires that a large fraction of *bandwidth* is controlled by honest parties. Our committee-election procedure provides security given only this trust assumption, and in particular does not require that a majority of relays are honest. Electing committee members with probability proportional to their bandwidth serves the dual purpose of allowing us to reason that even small committees must contain an honest party, despite the fact that a majority of relays may be malicious.

In addition, to make our protocols better suited to deployment in the Tor network, we develop new techniques for offline preprocessing and input sharing to make our protocols more resilient against party churn and malicious behavior. In particular, we allow some committees to fail during the preprocessing without interrupting the overall protocol execution. This is not necessary in prior work when only a single committee is elected, but becomes a requirement as we aim to better utilize bandwidth. Additionally, we achieve the following, seemingly contradictory properties for input providers using an *accountable* input protocol: (1) a malicious member of the committee receiving the inputs cannot exclude the input of an honest party, and (2) a malicious input party that is not on the committee cannot cause the secure computation to abort. These properties together allow statistics to be computed despite missing or malformed inputs without allowing the adversary to degrade privacy by selectively excluding all but a subset of targeted inputs.

We have implemented our protocol and experimentally demonstrate capabilities that exceed the limitations of existing proposals and Tor's current measurement methods. First, we show an example of using robust statistics by computing the median of the relays' inputs. This statistic tolerates outliers and thus prevents malicious Tor relays from manipulating measurement outcomes through spurious inputs. It also doesn't require us to know in advance what input values are reasonable, as in prior work using input validation [21]. Such robust aggregate statistics provide privacy while also providing outputs that can be relied on for network-critical operation, such as measuring bandwidth capacity [45].

We also demonstrate that Stormy can be used to compute efficient statistics based on sketches. These computations are not necessarily robust, but their space efficiency enables the collection of a variety of useful network statistics. We consider the *count distinct* computation and show how to count unique items across the entire network with *exponentially less* communication and computation than is typically required by protocols for private set intersection or union [31, 51, 61]. This design supports accurately counting distinct items up to the billions, allowing Tor to detect how many unique users it has, how many distinct Web domains its users visit, and how many of its onion services are visited at least once. An analysis of our design shows that both the median and count-distinct computations can be performed from dozens to hundreds of times per day, enabling Tor to collect and report these new statistics quickly and with regularity.

To summarize our contributions: (1) we describe two deployment models to incorporate MPC into the Tor network based only on standard Tor assumptions; (2) we develop techniques for providing input and offline processing that are resilient to party failures and prevent omission of inputs; (3) we adapt MPC protocols for non-uniform trust and bandwidth; (4) we describe how to securely compute a median and sketch-based unique count, which is not possible with current Tor-measurement systems; and (5) we provide

experimental results showing practical MPC performance when run over both a small set of authorities and the entire Tor network.

2 TOR BACKGROUND

Tor [26] anonymizes Internet traffic by sending it through its network of *relays*. The relays are run by independent volunteers who donate the computational and network resources [5]. A Tor user creates an anonymized TCP connection through Tor by sending a connection request to a locally-run Tor *client*. The client builds a *circuit* through a sequence of relays, and the desired connection can be placed onto that circuit. Tor circuits generally consist of three relays: a *guard*, *middle*, and *exit*. Relays are flagged for suitability as guards or exits based on their resources and willingness to connect outside the network, and then, for each position in a circuit, a relay is chosen from among those suitable with probability proportional to a network-determined *weight* [6]. That weight is intended to be proportional to the relay's bandwidth, largely to balance the traffic load and improve network performance. However, doing so also provides security: it requires an adversary to provide costly bandwidth to the network in order for its relays to achieve positions in which they can attack clients.

Tor users are vulnerable to an adversary that controls a significant fraction of the Tor network. For such an adversary, there is a non-trivial chance that a client's circuit is composed entirely of malicious relays, in which case the adversary can easily deanonymize the connection. In practice, however, the adversary need only control a circuit's first and last hop (i.e. the guard and exit) because he can identify that both are part of the same circuit by correlating traffic patterns [9, 17]. Controlling any one position harms client security as well, as a malicious guard can, for example, perform website fingerprinting [67] and selective denial-of-service [13], a malicious middle can perform website fingerprinting [43] and guard discovery [39], and a malicious exit can perform man-in-the-middle attacks [69]. Thus, for Tor to be secure it must be that no adversary controls a large fraction of the relay weight in any position. While no sharp threshold for security exists, an adversary that controlled, say, 25% of Tor's bandwidth, would effectively have compromised the network, as under current rates of churn, a quarter of clients could expect to choose a compromised guard immediately, and the rest within a few months; given a compromised guard, the client can expect to choose a compromised exit (and thus be deanonymized via a correlation attack) within hours [29, 47]. Tor's threat model is thus limited to an adversary controlling a small fraction of bandwidth (we will assume $< 25\%$).

The state of the Tor network is maintained by the Directory Authorities (DirAuths) [4]. There are currently nine DirAuths that vote to determine a network *consensus*. A consensus is produced every hour and contains, among other things, a list of the relays with their bandwidths and position flags. DirAuths also store relay *descriptors* that contain other data needed by clients, such as the exits' connection policies. Every client downloads a copy of the consensus every hour and downloads sufficiently-recent descriptors (currently within 18 hours), which it uses to choose relays when constructing circuits. Most entries in the consensus, including the relays and their properties, must be voted for by a majority of the DirAuths, and so Tor relies to a great extent on a trust assumption that a majority of DirAuths are honest. The DirAuths also generate a random value to put into the consensus using a commit-reveal protocol [7]. This value is currently only used to affect how Tor's internal name-resolution operates.

To be listed in the Tor consensus, a relay must directly communicate with the nine DirAuths and an additional set of Bandwidth Authorities who determine the relay's bandwidth and consensus weight. Authorities are geographically distributed around the world in many different networks; consequently, relays must be well-connected in the Internet in order to communicate with each authority. Moreover, the Tor protocol assumes that each relay can communicate with all other relays in a fully-connected network.

The Tor network consensus from 2018-10-01 includes 6,331 relays. On that day, Tor relays sent on average about 125 Gbps of traffic in aggregate on behalf of an estimated 2 million users. We observe that the distribution of Tor relay weight is skewed towards high-bandwidth nodes. The largest 25% of relays by weight have 78% of the total weight. The minimum non-zero advertised bandwidth is 0.02 Mbps, the median is 12.44 Mbps, and the maximum is 1,397 Mbps. The total advertised bandwidth is 275 Gbps. Much of Tor's bandwidth goes unused: the relays' bandwidth histories show that they actually relayed only 125.0 Gbps of traffic on average, and so only 45.5% of the advertised bandwidth is used. Moreover, we observe that 95% of Tor relays (by bandwidth) have at least 25% spare capacity (see the technical report [65] for more detail). This behavior is consistent over time, as we observe that for every day in 2018-10 at most 48.3% of Tor's advertised bandwidth is used.

3 SYSTEM MODELS

3.1 Deployment Models

Stormy, our system enabling Tor to securely measure and monitor itself, can be deployed in two models: (1) the **Authority Model** (**AuthMode**) in which a small set of authorities (e.g. the Directory Authorities) is dedicated to receiving inputs and performing the secure computation, and (2) the **Relay Model** (**RelMode**) in which Tor relays themselves are used to perform the secure computation. The advantage of the Authority Model is efficiency. The Relay Model has the advantage of not requiring bandwidth beyond that provided already by Tor. Additionally, the Relay Model will only use Tor's existing trust assumption that a large fraction of relays by weight is honest. In contrast, while the Directory Authorities are already entrusted with significant power, using them as computation parties would give them a new ability to *covertly* learn private information about the *past*. The Relay Model is also more consistent with designs that decentralize the functions of the DirAuths [55, 58, 59]. These reasons may justify using the Relay Model of Stormy despite its relative inefficiency.

3.2 Network and Adversary Model

We model the communication network and its hosts based on Tor as it currently exists. We assume the hosts participating in the protocols, the hosts' public keys, and the hosts' relay weights are publicly known and agreed upon (all this information is in the Tor consensus). We further assume that hosts communicate directly over confidential and authenticated channels, and that maximum delays between hosts on these channels are known.

The adversary we consider is malicious (i.e. active) and in control of some Tor hosts. In the Authority Model, we assume at least one

of the authorities is honest. This is similar to (and weaker than) the existing assumption of an honest majority of DirAuths. In the Relay Model, we instead assume that the adversary controls relays with at most 25% of total relay weight. As discussed in Section 2, this is a commonly-used limit on a reasonable Tor adversary, as it represents a basic security assumption in Tor. Also, without it, many of the inputs to Stormy would be observed before any secure computation even began. Note that we do not assume that a majority of relays (by number) is honest because this is neither necessary nor sufficient for the security of Tor itself. We allow the adversary to passively observe all of the communication channels between hosts (a threat that Tor itself is actually not secure against).

4 SECURE COMPUTATION PROTOCOLS

Our computation consists of several stages divided between offline and online phases. The offline phase includes those stages that can be completed before the inputs are known. Once the inputs are available, the online phase can begin. We describe the stages in the Relay Model, as the Authority Modelis a special case in which a single committee consisting of the authorities runs all components.

First, committees are elected to run different components of the system (Section 4.1). We sample large enough committees to ensure (with all but negligible probability) that each committee has at least one honest participant. Next, a designated committee generates secret-shared, authenticated, random bits, while the rest of the committees each run a protocol to generate secret-shared, authenticated, AND triples. These bits and triples will be used during the online phase (Section 4.2). An important feature of this process is that a triple-generating committee can abort (e.g., due to host failure) without requiring the other committees to abort. After enough bits and triples have been generated, the online phase begins when the relays' inputs are available. To start it, a designated committee executes the input-sharing protocol with each relay, receiving encoded inputs (Section 4.3). An important and novel feature of this protocol is that it does not allow a malicious input party to cause the overall computation to fail, while also preventing malicious committee members from excluding honest inputs. Finally, the same committee runs the computation protocol to evaluate a Boolean circuit on the supplied inputs (Section 4.4).

We will show that each protocol run by a committee is secure against a malicious adversary as long as at least one committee member is honest. Moreover, we prove in the Relay Model that the composed system is secure against an adversary that controls a fraction $f < 1$ of the total bandwidth. Composed security in the Authority Model holds following similar arguments assuming at least one authority is honest.

Our protocols follow the paradigm for secure computation of computing on authenticated shares. In the following protocol descriptions, we denote the global MAC key by Δ. We denote a value v that is additively secret-shared among a set of parties C as $[v]_C$; $v^{(i)}$ denotes P_i's share of $[v]_C$ for $P_i \in C$. We indicate a value v and its MAC $\mu = \Delta v$ that have been secret-shared among C as $[\![v]\!]_C$ (i.e. $[\![v]\!]_C = ([v]_C, [\mu]_C)$). We often omit the subscript when the set C is clear from context. We denote by $[\![x]\!]^{(i)}$ the shares party P_i has of x and its MAC μ, that is, $[\![x]\!]^{(i)} = \left(x^{(i)}, \mu^{(i)}\right)$. We use H to denote a cryptographic hash function. We use $x \leftarrow S$ to indicate that x is chosen uniformly at random from S. Finally, we let λ denote a statistical security parameter and κ denote a computational security parameter. These are set to 40 and 256 respectively in our experiments.

4.1 Committee Election

We describe how committees are elected in the Relay Model.

4.1.1 Generating Randomness. All parties begin by agreeing on a random string, which they will use to locally run the committee-assignment algorithm described below. To securely obtain unbiased random bits for committee election, we use the randomness already generated by the Directory Authorities and included in the consensuses [7]. The security of this randomness relies on the assumption that a majority of the DirAuths are honest. Using this assumption for secure computation on network data does increase the consequences of violating it, including in particular a new power to reveal information about the activity in the network from *before* the point of compromise. To avoid relying on the randomness generated by the Directory Authorities, the relays themselves may perform commit-reveal randomness generation using a consensus protocol suitable for large distributed systems (i.e. with low-communication complexity and responsiveness to latency [57, 71]).

4.1.2 Committee Assignments. We use the securely-generated shared randomness to elect a set of committees to perform the secure computation. Two types of committees are used: (1) Triple Committees (TCs) that generate AND triples during the offline phase, and (2) Computation Committees (CCs) that generate authenticated random bits offline and perform the online computation. All relays know the committees because they are generated locally from the shared randomness and using the same consensus document (see Section 5 for further discussion). The committees are used for all computations within a given time period, after which new committees are selected using fresh shared randomness.

We wish to choose committees such that, with probability at least $1 - 2^{-\lambda}$, all of them have at least one honest member. To accomplish this, we fix a committee size c and then select each committee independently by choosing c members at random (with replacement) with probability proportional to their consensus weights. Since we sample committee members with replacement, parties may be selected onto multiple committees, and the parties with more bandwidth will be assigned to more committees. We sample a large number m_{TC} of Triple Committees, many of which will be used in parallel to exploit Tor's available bandwidth. We also sample a smaller number m_{CC} of Computation Committees, of which we will only use one at a time, but keep several in reserve to recover from node failures. For now, we focus on the case where only one Computation Committee is used; see Section 5 for how multiple committees are used. Let $m = m_{\mathrm{TC}} + m_{\mathrm{CC}}$. Then, using Tor's security assumption that the adversary controls at most a small fraction f of the network bandwidth, we have the following claim.

Claim 1. *If m committees are sampled, and each is of size $c = \lceil(\lambda + \log_2(m))\,/\,\log_2(1/f)\rceil$, then the probability that some committee contains c malicious parties is at most $2^{-\lambda}$.*

Proof. Since each party is sampled independently, the probability that a committee of size c is entirely malicious is f^c. Using

Protocol Π_{Pre}

Notation:

- Let CC be the Computation Committee.
- Let $\mathsf{TC}_1, \ldots, \mathsf{TC}_{m_{\mathsf{TC}}}$ be the Triple Committees.

Initialize:

1. *Generate Δ and Δ_j*: Each $P_i \in$ CC chooses $\Delta^{(i)} \leftarrow \mathbb{F}_{2^\lambda}$, defining global MAC key $\Delta = \sum_{i=1}^{c} \Delta^{(i)}$. Each TC_j similarly generates $\Delta_j \leftarrow \mathbb{F}_{2^\kappa}$.
2. *Transfer Δ*: Each party $P_i \in$ CC does the following:
   a. For each TC_j and each $P_k \in \mathsf{TC}_j$, choose $\Delta^{(i)(k)} \leftarrow \mathbb{F}_{2^\lambda}$ s.t. for all j, $\sum_{P_k \in \mathsf{TC}_j} \Delta^{(i)(k)} = \Delta^{(i)}$.
   b. Send $\Delta^{(i)(k)}$ to P_k. Let $\Delta^{(k)} = \sum_{P_i \in \mathsf{CC}} \Delta^{(i)(k)}$ be the share of the MAC key Δ held by $P_k \in \mathsf{TC}_j$.

Random$(\mathbb{F}, b)$**:**

1. *Generate random bits*: If $\mathbb{F} = \mathbb{F}_2$, each $P_i \in$ CC calls $\Pi_{\mathsf{aShare}}(\Delta^{(i)}, b)$ and aborts if this call aborts. CC receives bits $([\![r_1]\!] \ldots [\![r_b]\!])$ as output.
2. *Generate random $\mathbb{F}_{2^\lambda}$ elements*: If $\mathbb{F} = \mathbb{F}_{2^\lambda}$, each $P_i \in$ CC calls $\Pi_{\mathsf{aShare}}(\Delta^{(i)}, b\lambda)$ and aborts if this call aborts. CC receives bits $([\![r_1]\!] \ldots [\![r_{b\lambda}]\!])$ as output and combines each consecutive λ bits to produce $\mathbb{F}_{2^\lambda}$ elements $([\![s_1]\!] \ldots [\![s_b]\!])$.

Triples(ℓ)**:** (Let TC_j be the TC calling the protocol.)

1. *Generate triples*: Each $P_i \in \mathsf{TC}_j$ runs $\Pi_{\mathsf{aAND}}(\Delta_j^{(i)}, \ell)$, and then, for each value in the resulting triples, P_i executes $\Pi_{\mathsf{MACSwitch}}$ with $\Delta_j^{(i)}$ and $\Delta^{(i)}$ and produces triple shares $([\![x_k]\!]^{(i)}, [\![y_k]\!]^{(i)}, [\![z_k]\!]^{(i)})$, $1 \le k \le \ell$. If the call to Π_{aAND} aborts, P_i informs each $P_j \in$ CC and aborts.
2. *Transfer triples*: For each triple component $[\![w]\!]^{(i)}$ held by a $P_i \in \mathsf{TC}_j$:
   a. P_i chooses $s_h^i \leftarrow \mathbb{F}_{2^\kappa}$, $1 \le h \le c-1$, sends $z_i = [\![w]\!]^{(i)} \bigoplus_{h=1}^{c-1} \mathrm{PRG}(s_h^i)$ to $Q_i \in$ CC, and sends each s_h^i to a distinct remaining $P_k \in$ CC.
   b. Each $P_k \in$ CC computes $[\![w]\!]^{(k)} = z_k \bigoplus_{h \ne k} \mathrm{PRG}(s_k^h)$.
3. *Check triples*: Each $P_i \in$ CC calls Π_{MACCheck} with Δ and the set of all triple component shares $[\![w]\!]^{(i)}$. If the call aborts, P_i informs each $P_k \in \mathsf{TC}_j$, causing each $P_k \in \mathsf{TC}_j$ to abort, and P_i rejects this and future triple transfers from TC_j.

Figure 1: Protocols for offline preprocessing.

a union bound over m committees, we require that $mf^c \le 2^{-\lambda}$. Solving for c yields the claim. ∎

4.2 Offline Preprocessing Protocols

The offline preprocessing protocols provide authenticated secret-shared random bits and AND triples. A random bit is needed for each input bit in the circuit, and a triple is needed for each AND gate in the circuit. Therefore, the preprocessing protocols can provide sufficient bits and triples for the online computation knowing only upper bounds on the number of input bits and the number of AND gates in the circuit.

The preprocessing protocols make use of the Π_{aShare} and Π_{aAND} protocols of Wang et al. (Figures 15 and 18 of [68], respectively). These protocols produce *pairwise* authenticated shared bits. We denote a bit x authenticated and shared in this way as $\langle x \rangle_{C,\Delta}$, where C is the group holding the shares, and $\Delta \in \mathbb{F}_{2^\kappa}$ is the MAC key used for authentication. We omit the C or Δ subscript when it is clear from the context. Under the pairwise authentication, the bit value is shared as $[x]_C$, and, for each $P_i \ne P_j$, P_i holds an authentication tag $\mathsf{M}_j[x^{(i)}] \in \mathbb{F}_{2^\kappa}$ on its share $x^{(i)}$ under a key $\mathsf{K}_j[x^{(i)}] \in \mathbb{F}_{2^\kappa}$ held by P_j. The key is uniformly random (i.e. $\mathsf{K}_j[x^{(i)}] \leftarrow \mathbb{F}_{2^\kappa}$), and the authentication tag is produced such that $\mathsf{M}_j[x^{(i)}] = \mathsf{K}_j[x^{(i)}] + x^{(i)}\Delta^{(j)}$. We denote the share of $\langle x \rangle$ held by P_i as $\langle x \rangle^{(i)}$, that is, $\langle x \rangle^{(i)} = \left(x^{(i)}, \{\mathsf{M}_j[x^{(i)}], \mathsf{K}_i[x^{(j)}]\}_{j \ne i}\right)$. A pairwise-authenticated value $\langle x \rangle$ can easily be turned into a globally-authenticated value $[\![x]\!]$ under the first λ bits of Δ. To do so, each party P_i sets its global MAC share $\mu^{(i)}$ to the first λ bits of $x^{(i)}\Delta^{(i)} + \sum_{j \ne i} (\mathsf{M}_j[x^{(i)}] + \mathsf{K}_i[x^{(j)}])$.

The preprocessing protocols are given as subprotocols of the combined preprocessing protocol Π_{Pre}, shown in Figure 1. Π_{Pre} is run by the Computation Committee (CC) and the Triple Committees $(\mathsf{TC}_1, \ldots, \mathsf{TC}_{m_{\mathsf{TC}}})$. We will show that Π_{Pre} realizes the functionality $\mathcal{F}_{\mathsf{Pre}}$ (Figure 6 in Appendix A). The Π_{Pre} subprotocols work as follows:

4.2.1 Initialize. CC initially generates the global MAC key Δ and distributes shares of it to each TC_i.

4.2.2 Random. This protocol is run by the CC to generate secret-shared, random, authenticated elements in $\mathbb{F}_2$ or $\mathbb{F}_{2^\lambda}$. It uses Π_{aShare} to generate random bits. To instead generate an element of $\mathbb{F}_{2^\lambda}$, it uses the technique of Keller et al. [49] to combine λ random bits. The protocol takes as input a field $\mathbb{F}$ and the number b of random field elements to produce. It outputs to CC the secret-shared authenticated random field elements $([\![r_1]\!]_{\mathsf{CC}}, \ldots, [\![r_b]\!]_{\mathsf{CC}})$.

4.2.3 Triples. The protocol is run by CC and a TC_j. It takes as input the number ℓ of triples to produce. It outputs to CC ℓ triples $([\![x]\!]_{\mathsf{CC}}, [\![y]\!]_{\mathsf{CC}}, [\![z]\!]_{\mathsf{CC}})$, where $x, y, z, \in \mathbb{F}_2$ and $x \wedge y = z$. TC_j runs the Π_{aAND} protocol to generate triples. However, our use of that protocol raises an issue that we must address. Specifically, Π_{aAND} allows a selective failure attack wherein $\mathcal{A}$ can learn a few bits of the MAC key Δ. Wang et al. can deal with this by using the randomness extraction technique of Nielsen et al. [60]. However, the situation is more challenging for us because we need to allow for offline committees to abort without halting the overall computation. If they all use the same global Δ, then each offline committee could attempt to learn some bits of Δ, and even if some are detected, the computation would proceed despite a large leakage of Δ overall.

We make an important addition to Π_{aAND} to allow a triple committee to abort without leaking any bits of the global MAC key Δ, thus allowing the other committees to continue. We achieve this by first having each TC_j generate authenticated triples using its own MAC key Δ_j. Then each committee calls $\Pi_{\mathsf{MACSwitch}}$ (Figure 2) on each triple to change the authentication tags to be under the global Δ. This protocol uses the MAC switch technique of Wang et al. [68] (Step 5, Figure 8 of [68]). Because different keys are used by each committee, the leakage cannot accumulate across committees, and so the randomness extraction in Π_{aAND} prevents leaking any bits of Δ. If the Π_{aAND} call aborts, each member of TC_j informs each member of CC, and then TC_j aborts.

Protocol $\Pi_{\text{MACSwitch}}$

Notation:

- Let $C = \{P_1, \ldots, P_c\}$ be the committee executing the protocol.
- $[\Delta_1]_C \in \mathbb{F}_{2^\kappa}$ is input as the current MAC key.
- $[\Delta_2]_C \in \mathbb{F}_{2^\lambda}$ is input as the desired MAC key.
- $\langle x \rangle_{C,\Delta_1}$, $x \in \mathbb{F}_2$, is input as the value on which to perform the MAC switch.

Protocol:

1. For each $P_i, P_j \in C, P_i \neq P_j$:
   a. P_i computes $\mathsf{K}'_i[x^{(j)}] = H(\mathsf{K}_i[x^{(j)}])$ and $U_{i,j} = H(\mathsf{K}_i[x^{(j)}] \oplus \Delta_1^{(i)}) \oplus \mathsf{K}'_i[x^{(j)}] \oplus \Delta_2^{(i)}$.
   b. P_i sends $U_{i,j}$ to P_j.
2. P_j computes $\mathsf{M}'_i[x^{(j)}] = x^{(j)} U_{i,j} \oplus H(\mathsf{M}_i[x^{(j)}])$.
3. The output to $P_i \in C$ is $\langle x \rangle^{(i)}_{C,\Delta_2} = (x^{(i)}, \{\mathsf{K}'_i[x^{(j)}], \mathsf{M}'_j[x^{(i)}]\}_{j \neq i})$.

Figure 2: MAC switching protocol.

After TC_j generates the triples, it transfers them to CC to use during the online computation. Each member $P_i \in \mathsf{TC}_j$ secret-shares $[\![w]\!]^{(i)}$ to CC, where w is a triple component, using $c-1$ seeds to a pseudorandom generator (PRG) to minimize communication. That is, P_i samples $c-1$ PRG seeds, $s_1^i, \ldots s_{c-1}^i$, and sends each to a different member of $\mathsf{CC} \backslash Q_i$, where $Q_i \in \mathsf{CC}$ is the ith member of CC. To Q_i, P_i sends $[\![w]\!]^{(i)} \bigoplus_{h=1}^{c-1} \mathrm{PRG}(s_h^i)$. Having each $P_i \in \mathsf{TC}$ send this to a different $Q_i \in \mathsf{CC}$ provides load balancing. After CC has received the triples from TC_j, it must ensure that they are well formed so that a malicious member of TC_j can't cause the later online computation to abort. CC executes a batch MAC check on all triple components using Π_{MACCheck} (Figure 3). This MAC-checking procedure is taken from Keller et al. [49]. If the MAC check fails, CC informs TC_j, TC_j aborts, and CC rejects the triple batch and any further batches from TC_j.

4.2.4 Security. Theorem 4.1 shows that Π_{Pre} securely realizes $\mathcal{F}_{\mathsf{Pre}}$ (Figure 6 in Appendix A) as long as there exists at least one honest member in every committee. The proof appears in the technical report [65].

Theorem 4.1. *Π_{Pre} realizes $\mathcal{F}_{\mathsf{Pre}}$ in the standalone model with random oracle H against a static, malicious adversary simultaneously corrupting up to $c-1$ members of each of CC, TC_1,.., and $\mathsf{TC}_{m_{TC}}$.*

A consequence of Theorem 4.1 (following from the definition of $\mathcal{F}_{\mathsf{Pre}}$) is that Π_{Pre} can be used to obtain random authenticated bits and triples secret-shared by CC that are unknown to the adversary and have correct MAC tags under a global key Δ. An additional corollary is that TC_j can only cause its triple generation to be aborted and cannot interrupt triple generation by other TCs. Furthermore, Π_{Pre} reveals no information about Δ to the adversary.

4.2.5 Performance. The offline preprocessing protocols are the most costly components of Stormy. Generating ℓ triples involves an execution of a correlated oblivious transfer with errors protocol (Figure 19 in [48]) with *every other* committee member. This protocol extends a small number of "base" 1-of-2 oblivious transfers [18] (which are the only offline asymmetric-key cryptographic

Protocol Π_{MACCheck}

Committee C uses shared key $[\Delta]$ to check the MACs of authenticated secret-shared bits $\{[\![x_h]\!]\}_{h=1}^b$:

1. C calls Π_{Pre}.Random($\mathbb{F}_{2^\lambda}$, 1) to obtain authenticated $\mathbb{F}_{2^\lambda}$ element $[\![f]\!]$.
2. *C generates random coefficients*:
   a. Each P_i chooses $s_i, r_i \in \{0,1\}^\kappa$ and sends commitment $\mathrm{Com}_i = H(s_i || r_i)$ to each P_j.
   b. After all Com_j are received, each P_i sends opening $(s_i || r_i)$ to each P_j.
   c. After obtaining all opened values, each P_i sets $r = \bigoplus_j r_j$ and generates $(g_0, g_1, \ldots, g_b) \leftarrow \mathrm{PRG}_\lambda(r)$, $g_h \in \mathbb{F}_{2^\lambda}$ for $0 \leq h \leq b$.
3. Each P_i computes $y^{(i)} = f^{(i)} g_0 + \sum_{h=1}^b x_h^{(i)} g_h$ and $\mu^{(i)} = \mu_f^{(i)} g_0 + \sum_{h=1}^b \mu_{x_h}^{(i)} g_h$, where $\mu_z^{(i)}$ denotes P_i's share of the MAC in $[\![z]\!]$.
4. *C performs a partial opening*:
   a. Each P_i sends $y^{(i)}$ to a designated party P_1.
   b. P_1 sends $y = \sum_{j=1}^c y^{(j)}$ to each P_i.
5. Each P_i computes $\zeta^{(i)} = \mu^{(i)} - y\Delta^{(i)}$, chooses $q_i \in \{0,1\}^\kappa$, and sends commitment $\mathrm{Com}_i = H(q_i || \zeta^{(i)})$ to each P_j.
6. After all Com_j are received, each P_i sends opening $(q_i || \zeta^{(i)})$ to each P_j.
7. After obtaining all opened values, each P_i sets $\zeta = \sum_{j=1}^c \zeta^{(j)}$ and aborts if $\zeta \neq 0$.

Figure 3: MAC checking protocol.

operations) into a large number (ℓ) of oblivious transfers using only symmetric key operations. However, it has high communication costs, and with c parties, it requires that each party sends approximately $3(c-1)(\kappa+\lambda)\ell\beta$ bits for ℓ triples, where $\beta \leq \lambda / (\log_2(\ell)) + 1$. The Random functionality is also based on OT extension and thus also requires relatively cheap computation, but its communication costs for b bits are about $2(c-1)\lambda b$ and are thus significantly lower than triple generation unless $b \gg \ell$.

4.3 Input Sharing Protocol

Another critical part of our protocol is in how inputs are provided to the Computation Committee, CC. Specifically, our input protocol must (1) ensure that a malicious committee member cannot modify or exclude the input of an honest party and (2) prevent a malicious input party from causing the computation to abort. To reduce the amount of time input parties must be online, we also want this protocol to be "non-interactive" in the sense that there is only one message sent from an input party to the committee (further interaction within the committee is allowed).

One of the challenges in achieving these properties simultaneously is the need for parties to prove that they did not receive a message that should have been sent. To support such proofs, we design the accountable message functionality $\mathcal{F}_{\mathsf{AccMsg}}$, described in Figure 8 (Appendix A). The functionality has send and reveal subroutines. The send subroutine delivers a message to the receiving party while allowing him to prove to the other committee members if he failed to receive a message. The reveal subroutine simply forwards a sent message to all committee members. We describe in the technical report [65] how to realize $\mathcal{F}_{\mathsf{AccMsg}}$ using an encryption

scheme with verifiable decryption (e.g. El Gamal). It requires $O(c)$ communication, with the sender sending an encrypted message to every committee member, each of which forwards the ciphertext to the receiver.[1]

We use $\mathcal{F}_{\mathsf{AccMsg}}$ for sending point-to-point messages of both public values and c-out-of-c additive shares. In the latter case, the functionality looks a lot like weak verifiable secret sharing (WSS) [62], which guarantees agreement on an honest dealer's shared value, and allows disagreement on whether to abort when the dealer is malicious.[2] However, there are a few important differences. We require an additive sharing of the input (rather than, say, a Shamir sharing), and we do not want to involve the dealer after the sending phase. It is not clear how to achieve these properties using WSS. Note that, because we use an additive sharing, reconstruction is *always* possible, as long as everyone has received some signed value in the field. Additionally, there are a few relaxations that we leverage: we allow some honest parties to abort, even when the dealer is honest (as long as they don't blame the dealer), and we allow disagreement on the dealer's input value if the dealer is malicious.

We now briefly describe the Π_{Input} protocol (Figure 4) making use of $\mathcal{F}_{\mathsf{AccMsg}}$. A functionality $\mathcal{F}_{\mathsf{Input}}$ is given in Figure 7 (Appendix A) and a proof of security for this protocol is given in the technical report [65]. P_{in} sends additive shares of his masked, b-bit input, and of the b mask values. He cannot authenticate these values, since he does not know Δ, so he instead computes part of the MAC check protocol. Specifically, he secret-shares random coefficients $(\chi_1, \ldots, \chi_b)$, and computes and sends their inner-product with his mask values.

The committee transfers the MAC values from the pre-processed random bits to the masked input, and then opens the shared coefficients in order to complete the MAC check on the masked input. If the MAC check terminates without error, they know they have validly authenticated, unmodified input. If an error is detected during the MAC check, there are two possibilities: either the input party sent an inner-product that was inconsistent with his mask values and his MAC check coefficients, or some committee member modified some of the values he received. The committee members use the reveal subroutine of $\mathcal{F}_{\mathsf{AccMsg}}$ to verify that the masks, the coefficients, and the inner-product are consistent; crucially, they can do this without exposing the masked input, and so an honest P_{in} is not adversely impacted.

The input protocol has different abort behavior based on which party is controlled by the adversary. If only P_{in} is malicious, we guarantee that all committee members detect this and blame P_{in}, so they can exclude his input and continue. If only committee members are malicious, some honest parties may not immediately detect this, but none will blame P_{in}. We allow the protocol to continue with a fraction of the honest parties, who will quickly abort anyway, as at least one of their honest partners will have stopped participating. Finally, if both the input party and committee members are malicious, they can force part of the committee to blame the input party while others blame the committee (or nobody at all). Regardless, all committee members will terminate the protocol, either immediately, or when detecting that others have aborted. The proof of the next theorem appears in the technical report [65].

Theorem 4.2. *Π_{Input} securely realizes the ideal functionality $\mathcal{F}_{\mathsf{Input}}$ in the standalone model against a static, malicious adversary that can corrupt P_{in} and at most $c-1$ out of the c parties in CC.*

4.4 Online Computation Protocol

The online computation protocol is run by the Computation Committee, CC, to evaluate a Boolean circuit on inputs provided by the relays. We use the TinyOT protocol of Burra et al. (Figure 11, [15]). The online computation protocol Π_{Online} uses the subprotocols Π_{Add}, Π_{Multiply}, and Π_{Output} described by Burra et al. It takes as input the circuit $\mathbb{C}$ to be computed and the global MAC key $[\Delta]$. Access to the offline bits and triples is assumed as well. After running Π_{Input} with each input party, CC executes Π_{Add} and Π_{Multiply} for each of the XOR and AND gates in $\mathbb{C}$, respectively, in topologically-sorted order. The inputs at each gate are values $[\![x]\!]_{\mathsf{CC}}$ and $[\![y]\!]_{\mathsf{CC}}$ either obtained during input sharing or output by a previous gate evaluation, and the gate's output is a value $[\![z]\!]_{\mathsf{CC}}$, where $z = x \oplus y$ for an XOR gate and $z = x \wedge y$ for an AND gate. Π_{Output} takes the values of the o output gates $([\![y_1]\!]_{\mathsf{CC}}, \ldots, [\![y_o]\!]_{\mathsf{CC}})$ and either aborts or outputs $(y_1, \ldots, y_o)$ to each member of CC. The functionality $\mathcal{F}_{\mathsf{Online}}$ realized by this protocol is given in Figure 9 (Appendix A), and the following theorem follows from Burra et al. [15].

Theorem 4.3. *Π_{Online} securely realizes the ideal functionality $\mathcal{F}_{\mathsf{Online}}$ in the standalone model against a static, malicious adversary that can corrupt at most $c-1$ out of the c parties in CC.*

The online computation phase is fast and inexpensive compared to the offline preprocessing phase. It involves no asymmetric-key operations. Evaluating XOR gates is local—no network communication occurs. Evaluating an AND gate involves sending and receiving about two bits on average. A major cost is latency due to the round complexity, which is determined by the circuit's AND-gate depth.

4.5 Complete protocol

We now describe how these component protocols can be assembled into a protocol $\Pi_{\mathsf{RM-MPC}}$ to securely evaluate a Boolean circuit $\mathbb{C}$ in the Relay Model. The protocol in the Authority Model can be recovered by skipping committee election and executing everything in the single committee, and security follows in a straightforward way from prior work. $\Pi_{\mathsf{RM-MPC}}$ is executed as follows:

(1) All parties run the randomness generation protocol described in Section 4.1.1 to generate shared randomness r.
(2) Each party locally uses r to elect m committees, setting the first committee as the CC, and the rest as TCs.
(3) The CC runs $\mathcal{F}_{\mathsf{Pre}}$.Initialize to generate a global MAC key Δ.
(4) The CC runs $\mathcal{F}_{\mathsf{Pre}}$.Random to generate sufficiently many random bits and field elements for all inputs in $\mathbb{C}$.
(5) Each TC_j runs $\mathcal{F}_{\mathsf{Pre}}$.Triples to generate AND triples and outputs them to the CC. If any party in any TC_j aborts, no triples are produced, but the protocol continues.

[1] If we are willing to sacrifice the non-interactive property, the sender can send a message directly to the receiver without accountability, saving on communication. The receiver may then complain and ask for an execution of the accountable message protocol only when necessary. If everyone is honest, the protocol remains non-interactive.
[2] Because we sometimes use this functionality for sending public values, we model it as having send and reveal subroutines, instead of share and reconstruct, which is used in VSS and WSS.

Protocol Π_{Input}

Notation:

- Let $C = \{P_1, \ldots, P_c\}$ denote the Computation Committee holding global MAC key Δ.
- Let P_{in} denote the input party holding input bits $x_1, \ldots, x_b$.

Subroutine `Adjust`$([\![r]\!], [s]) \mapsto [\![s]\!]$**:** *($r, s \in \mathbb{F}_2$ or $r, s \in \mathbb{F}_{2^\lambda}$)*

1. Let $r^{(i)}$, $m_r^{(i)}$, and $s^{(i)}$ denote P_i's share of r, m_r (the MAC on r) and s, respectively.
2. Each P_i computes $d^{(i)} = s^{(i)} - r^{(i)}$. C opens d.
3. Each P_i sets $m_s^{(i)} = m_r^{(i)} + d \cdot \Delta^{(i)}$. C now holds $[\![s]\!]$.

Preprocessing:

1. C calls $\mathcal{F}_{\mathsf{Pre}}$(random, $\mathbb{F}_{2^\lambda}$, 1) to obtain authenticated, secret-shared field element $[\![r_0]\!]$.
2. C calls $\mathcal{F}_{\mathsf{Pre}}$(random, $\mathbb{F}_2$, b) to obtain authenticated, secret-shared bits $\{[\![r_h]\!]\}_{h=1}^{b}$.

Input Sharing:

1. P_{in} samples $s_0 \leftarrow \mathbb{F}_{2^\lambda}$, $(s_1, \ldots, s_b) \leftarrow \mathbb{F}_2^b$, and $(\chi_0, \ldots, \chi_b) \leftarrow \mathbb{F}_{2^\lambda}^{b+1}$. P_{in} computes $y = \sum_{h=0}^{b} (\chi_h \cdot s_h)$.
2. P_{in} forms random sharings $\{[s_h]\}_{h=0}^{b}$, $\{[x_h + s_h]\}_{h=1}^{b}$, and $\{[\chi_h]\}_{h=0}^{b}$. P_{in} accountably sends to $P_i \in C$ its shares, and the public value y, using $\mathcal{F}_{\mathsf{AccMsg}}$.send. If P_i receives (abort, C) in any execution of $\mathcal{F}_{\mathsf{AccMsg}}$, he aborts and blames C. Otherwise, if he receives (abort, P_{in}) in some execution, he excludes P_{in}'s input.
3. C calls `Adjust`$([\![r_h]\!], [s_h])$ and obtains $\{[\![s_h]\!]\}_{h=0}^{b}$.
4. Each $P_i \in C$ opens their share of $\chi_h^{(i)}$, for $h = 0, \ldots, b$, by calling $\mathcal{F}_{\mathsf{AccMsg}}$.reveal$(\chi_h^{(i)})$.
   (Here, and any other time $\mathcal{F}_{\mathsf{AccMsg}}$.reveal is called, if a party receives (abort, C), they abort and blame C.)
5. Each P_i computes $\chi_h = \sum_{i=1}^{c} \chi_h^{(i)}$, for $h = 0, \ldots, b$, and $m^{(i)} = \sum_{h=0}^{b} \left(\chi_h \cdot m_h^{(i)}\right)$, where $m_h^{(i)}$ denotes P_i's MAC share on s_h.
6. C executes a MAC check:
   a. Each P_i computes $\zeta^{(i)} = m^{(i)} - y \cdot \Delta^{(i)}$.
   b. C securely opens $\zeta = \sum_{i=1}^{c} \zeta^{(i)}$: each P_i sends a commitment $\text{Com}_i = H(\zeta^{(i)} || r_i)$ to all $P_j \in C$, and after receiving all $c - 1$ commitments, sends the opening $(\zeta^{(i)} || r_i)$ to C.
   c. If $\zeta \neq 0$, C must decide to exclude or abort:
      i. Each P_i echos his shares of s, $\{s_h^{(i)}\}_{h=0}^{b}$, by calling $\mathcal{F}_{\mathsf{AccMsg}}$.reveal$(s_h^{(i)})$.
      ii. Each P_i echos y by calling $\mathcal{F}_{\mathsf{AccMsg}}$.reveal$(y)$. If C detects any inconsistency, C excludes P_{in}'s input.
      iii. Each P_i computes $s_h = \sum_{i=1}^{c} s_h^{(i)}$. If $y \neq \sum_{h=0}^{b} (\chi_h \cdot s_h)$, C excludes P_{in}'s input. If not, C aborts and blames C.
7. Each P_i echos $\{(x_h^{(i)} + s_h^{(i)})\}_{h=1}^{b}$ by calling $\mathcal{F}_{\mathsf{AccMsg}}$.reveal$((x_h^{(i)} + s_h^{(i)}))$.
8. C locally computes $[\![s_h]\!] + (x_h + s_h) = [\![x_h]\!]$ for $1 \leq h \leq b$.

Figure 4: Protocol for input sharing.

(6) Each relay runs $\mathcal{F}_{\mathsf{Input}}$ with the CC to provide its input. If an input relay is disqualified, his input is dropped but the protocol continues. If a CC member aborts, the protocol stops.

(7) The CC runs $\mathcal{F}_{\mathsf{Online}}$ to securely evaluate the circuit $\mathbb{C}$. If any party in the CC aborts, the protocol stops.

To analyze the security of this protocol, we introduce a new model of security for Relay Model MPC protocols ($\mathcal{F}_{\mathsf{RM-MPC}}$) that we believe captures the requirements for MPC protocols run between a large number of parties. Specifically, our model builds on the *secure computation with abort and no fairness* definition given by Goldwasser and Lindell (Def. 5, [34]). This is a relaxation of security with abort in which the adversary is allowed to specify which honest parties abort and which ones receive output.

In our model we additionally designate a small set of parties (the CC) such that an adversary $\mathcal{A}$ is only allowed to abort the functionality (as described above) if CC $\cap\, \mathcal{A} \neq \emptyset$. This is necessary in a large-scale deployment where many more parties may participate in the protocol, but should not be able to interrupt the computation (either due to churn or malicious failure). Informally, the functionality provides security with abort against members of CC and full security (without allowing abort) against everyone else. A formal description of $\mathcal{F}_{\mathsf{RM-MPC}}$ is given in Figure 10 (Appendix A) and a proof of the following theorem is given in the technical report [65].

Theorem 4.4. *If m committees of size $c = \lceil(\lambda + \log_2(m)) / \log_2(1/f)\rceil$ are sampled, then $\Pi_{\mathsf{RM-MPC}}$ securely realizes $\mathcal{F}_{\mathsf{RM-MPC}}$ in a standalone $(\mathcal{F}_{\mathsf{Pre}}, \mathcal{F}_{\mathsf{Input}}, \mathcal{F}_{\mathsf{Online}})$-hybrid model against a static, malicious adversary corrupting less than an f fraction of the total available bandwidth.*

5 HANDLING PARTY CHURN

A key goal of Stormy is to be resilient to party failures as well as malicious behavior. We now discuss how we use the protocols described in Section 4 to achieve this. The discussion in this section is restricted to the Relay Model.

In Stormy, time is partitioned into a series of *epochs* (e.g. 24 hour periods). During an epoch, committees are elected once (when in the Relay Model), and then many rounds of offline preprocessing and online computation are run within the epoch. We wish to provide security over an epoch; that is, we bound the probability that the adversary succeeds in subverting security in committee election, offline preprocessing, or online computations to be at most $2^{-\lambda}$ during an epoch. In this section, we describe how Stormy operates over an epoch.

5.1 Committee Usage

In the Relay Model, two hours before the start of the epoch, Tor's DirAuths create an *Epoch Document*. This document is produced and distributed using Tor's existing consensus mechanism (Section 2) and is an extension of the existing daily generaton of shared randomness. Each relay downloads the Epoch Document in addition to the current consensus, and can be expected to be able to obtain it in time as relays already must have a consenus within two hours of the current time. The Epoch Document contains: (1) the random string currently produced daily by Tor, and (2) a list of relays and their consensus weights. The Epoch Document provides a consistent view of the network for a given epoch that each relay uses to locally determine the Triple Committees and Computation Committees. An Epoch Document is valid for only its epoch.

During the committee-election process in the Relay Model, we select m_{TC} Triple Committees and m_{CC} Computation Committees (see Sec. 4.1). However, we do not use all of these simultaneously; we find subsets with good performance, and hold some committees in reserve to replace committees that die due to churn. We consider every committee elected in an epoch to be in one of three states: *active*, *inactive*, and *dead*. Active committees are actively running a system protocol. Inactive committees are still responsive and available for use but have not yet been made active. Dead committees failed or aborted at some point in the epoch, and they are not used. All committees begin the epoch as inactive, some initial subset is made active, and, as active committees die, inactive ones may become active to replace them.

A main constraint on the use of TC committees is that we wish to limit Stormy's bandwidth consumption to a fraction b of Tor's total bandwidth. This reserves Tor's resources for its primary purpose of relaying client traffic. We set $b = 0.25$, leaving room for variation in Tor's traffic, as one half of Tor's bandwidth goes unused and 95% of relay bandwidth has at least 25% spare capacity.

The other main constraint on TC is the limited memory of the relays. A single relay may belong to several active TCs and thus simultaneously run many protocol executions. Each execution uses a non-trivial amount of memory (e.g. 291 MiB in the setting we will consider). Therefore, we must limit the number of active committees that share a single member.

We initially activate a TC if doing so doesn't violate these constraints, and increases overall bandwidth. That is, we consider the TCs in order and add them if the total bandwidth used is less than a b-fraction of Tor's total bandwidth, if no relay would use too much memory (we can use a conservative limit of 8 GiB for the largest 1% of relays and 3 GiB for the rest), and if no member of the committee has its bandwidth fully allocated to already-active committees. Then, when a committee dies during the epoch, we repeat the process with the remaining inactive committees.

We use a different process to activate a CC, because only one is active at a time. This committee has sole responsibility for bit generation and online computation, and thus should have high bandwidth. Therefore, we simply activate the inactive committee with the highest bandwidth. The same process applies if the active CC dies during the epoch.

5.2 Protocol Aborts

Unlike the set of stable authorities, temporary relay downtime is commonplace for benign reasons; for example, a relay operator may take his relay down to apply software patches, or a hosting center may lose power. To account for this natural churn, a design goal for Stormy is that the system should tolerate some failures instead of completely halting for the epoch when a relay goes offline or causes an abort. During triple generation, if a TC fails, each member of the TC notifies each member of the active CC, and the TC is marked dead. If there exist any inactive TCs, a new TC will be activated. The CC runs a MAC check on each batch of triples it receives from a TC to ensure that no errors have been introduced during transfer; if the MAC check fails, the TC is marked dead, and a new TC is activated if possible. The only other possible failures occur during bit generation, input sharing, and online computation—all within the active CC. If the active CC fails, each CC member notifies each relay in the network, and a new CC is activated if there are any inactive CCs. Stormy halts in RelMode only when all TCs or CCs are marked dead. Committees are reelected and protocols are restarted at the beginning of the next epoch.

5.3 Security Parameters

Among the system components, many of the potential security failure events occur independently. During committee election, there is a chance that some committee is composed entirely of malicious parties. There is a chance that a given triple committee deviates from the protocol and is not caught. Finally, there is a chance that the computation committee acts maliciously during bit generation, input sharing, or online computation and is not caught. To achieve λ-bit statistical security *overall* for an epoch, we must consider the probability that any one of these occurs.

In the Authority Model, setting all statistical security parameters to λ is sufficient regardless of the number of authorities, because there is no committee election, and all computation halts for the epoch upon any abort. The same is not true of the Relay Model — m_{TC} TCs and m_{CC} CCs are elected, and the adversary can attempt to cheat in each of them. The chance that a given committee is entirely malicious is f^c, where f denotes the adversary's fraction of bandwidth, and c denotes the size of each committee. Let $m = m_{TC} + m_{CC}$, let λ_1 be the statistical security parameter during triple generation, and let λ_2 be statistical security parameter during bit generation, input sharing, and online computation. By applying a union bound, the adversary's overall success probability is at most $mf^c + m_{TC}2^{-\lambda_1} + m_{CC}2^{-\lambda_2}$. To bound this by $2^{-\lambda}$, we simplify the security constraint by setting $\lambda_1 = \lambda_2 = \lambda'$, which yields the requirement that $m(f^c + 2^{-\lambda'}) \leq 2^{-\lambda}$. With $m = 1008$, $f = 0.25$, and $c = 25$, setting $\lambda' = 56$ is sufficiently large to provide overall $\lambda = 40$-bit statistical security.

6 TOR COMPUTATIONS

We present two computations that are useful in Tor, median and set-union cardinality, and we describe how they can be efficiently computed via MPC. Even just these two functions could be applied to measure and monitor many types of activity on the Tor network. Moreover, they demonstrate key techniques, such as sorting and sketching, needed for other types of computations. We expect that these methods could be applied to a wide variety of use cases within Tor, including for denial-of-service detection and mitigation, monitoring for protocol anomalies, detecting network errors and failures, understanding user behavior, tracking performance characteristics, and detecting blocking of clients and exits.

6.1 Median

The first function we demonstrate is the median of the relay inputs. The median is a robust statistic, insensitive to the presence of outliers. This property is valuable in the context of Tor because malicious relays can provide arbitrary inputs that might make other statistics, such as average or maximum, meaningless. For an example of how we might use median to to securely determine the number of circuit failures in a day, each relay can count the number it observes, then it can infer a global count by dividing by the fraction of circuits its sees (i.e. its bandwidth fraction), and then the

Table 1: Median circuit properties with 32-bit inputs.

# Inputs	1,000	3,000	5,000	7,000
# Input Bits	32,000	96,000	160,000	224,000
# Gates ($\times 10^6$)	6.70	2.76	54.0	78.4
# AND Gates ($\times 10^6$)	1.51	6.19	12.1	17.6
Depth	7,031	9,973	11,636	11,636
AND Depth	1,815	2,574	3,003	3,003

median of these values provides a robust estimate of the true total count. This method can be used for any measurement for which a relay can use its local measurements to make an inference of the global statistic, such as counting the number of circuits, bytes, clients, etc.

To securely compute the median, we use a circuit that sorts the input values and then outputs the middle one. We use a Batcher odd-even mergesort, which is a practical sorting network for realistic numbers of inputs [66]. For each compare-and-swap operation in the network, we use a comparison circuit with low AND-gate complexity [52]. The total number of AND gates in the resulting circuit on n inputs of b bits each is at most $bn\lceil\log_2^2(n)\rceil/2$, and its AND depth is $(b+1)(\lceil\log_2(n)+1\rceil\lceil\log_2(n)\rceil/2)$. Table 1 shows the size and depth of the median circuit for different numbers of 32-bit inputs. Using 32-bit inputs enables integer input values up to 4 billion, which is sufficient for most measurements of the Tor network that relays might make.

6.2 Set-Union Cardinality

The second function we demonstrate is the cardinality of the union of sets observed at the relays. That is, this function counts the number of distinct items among all items seen by relays. This computation is not robust, but could still provide much useful information about the Tor network, such as how many unique users it has, how that population changes over time, and how many different domains are visited.

Computing set-union cardinality is straightforward if the domain is small—each relay maintains its set as a bit vector, each observed item is hashed to an entry, and its value set to 1. The relays use the vectors as input to a secure computation of the OR of each entry, receiving as output the total number of entries with value 1. However, this approach doesn't scale well with the domain. Tor has estimated as many as 4 million different users in a day. Taking the union (i.e. the OR) of million-bit inputs from thousands of relays would require billions of expensive triples to be generated offline. Similarly, counting billions of distinct items (e.g. URLs visited) would require billions of additions and thus triples.

Therefore, we instead use a representation of set cardinalities that is much smaller and enables cheaper MPC through the use of free-XOR. Each relay stores a LogLog sketch [27], which provides space-efficient counting of distinct items given a fixed relative error. We choose LogLog over more-recent improvements [37] because the circuit computing the count of a LogLog sketch is simpler. Each LogLog sketch consists of k counters of bit-width w. We modify the standard sketch by (1) storing each counter in a unary representation (making the maximum stored value 2^w instead of 2^{2^w}), and (2) representing the 0 or 1 value at each counter entry with a bitstring of length s, where 0^s represents 0 and any non-zero string represents 1. These changes will reduce the number of expensive AND gates needed during the MPC computation.

A relay locally updates its LogLog sketch when it observes an item x. The relay hashes x to $H(x)$ and uses the first $\log_2(k)$ bits of $H(x)$ to determine which counter, C_i, to update. Examining the remaining bits of $H(x)$, the relay determines the largest number j of consecutive ones at the beginning of that bitstring. The relay then sets the first j entries of the chosen counter C_i to 1. The s-bit representation of each of those entries is set to 1 by choosing each of its s bits uniformly at random. We set $s \geq \lceil\lambda + \log_2(kw)\rceil$ so that, over all counter entries, a random value results in a non-zero value with probability at least $1 - 2^{-\lambda}$.

Using this input representation significantly improves the efficiency of the computation. It reduces the OR of the counter bits to XOR operations, which are free. This opens an attack in which a malicious Computation Committee member sets a counter to 0, but doing so requires guessing an unknown random s-bit value, and s is chosen so that this occurs with probability at most $2^{-\lambda}$. (Note that a CC member can easily change a 0 value to a 1, but this is allowed, since it can provide a logical 1 in its own input for any counter entry, which remains 1 after taking the union.) Therefore, we do not need to protect against errors until after input sharing and union are computed through bitwise XOR—input parties simply secret-share their inputs to the CC using $\mathcal{F}_{\text{AccMsg}}$, and committee members then just XOR the inputs. Only afterwards do they use the preprocessed random bits to add MAC tags to the resulting secret-shared values. The unique count is then computed from those aggregated counters.

To obtain this cardinality, the secure computation determines the last index z_i of a 1 entry in each counter C_i (0 if none) and returns the sum $z = \sum_i z_i$ of these indices. Durand and Flajolet [27] show that $\alpha k 2^{z/k}$ is an unbiased estimator of the true distinct count, where α is a constant that adjusts for bias. They also show that it has a standard error of approximately $1.3/\sqrt{k}$. Thus the value produced by the secure computation can be transformed into a cardinality estimate using public information.

The total number of AND gates in this LogLog circuit is at most $k(w(s+1) + \lceil\log_2(w)\rceil + \lceil\log_2(k)\rceil)$. Its AND depth is at most $\lceil\log_2(s)\rceil + w + \lceil\log_2(k)\rceil(\lceil\log_2(w)\rceil + \lceil\log_2(k)\rceil)$. Table 2 shows the size and accuracy of the LogLog circuit with counters of width $w = 32$ and counter entries of size $s = 55$ bits. With 32 entries per counter, cardinalities above 4 billion can be measured. For $k = 1,024$, setting the bits per entry at $s = 55$ yields the desired failure probability of at most 2^{-40}. Note the circuit size doesn't vary with the number of inputs, which means that the amount of MPC communication after input sharing would not be affected by growth in the Tor network.

7 EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of Stormy. We have implemented all of the protocols in roughly 10k lines of C/C++. [3] We use cryptographic primitives provided by the OpenSSL, Sodium, SimpleOT, and MIRACL libraries [1–3, 18]. SIMD CPU instructions are used in critical regions of code to improve performance. Our

[3] The code is available online at <https://github.com/rwails/stormy>.

Table 2: LogLog circuit properties after XORed relay inputs.

# Counters	128	512	1,024	4,096	8,192
# Input Bits ($\times 10^6$)	0.23	0.90	1.80	7.21	14.4
# Gates ($\times 10^6$)	0.47	1.89	3.78	15.1	30.2
# AND Gates ($\times 10^6$)	0.23	0.90	1.81	7.2	14.5
Depth	87	97	102	112	117
AND Depth	49	51	52	54	55
Std. Error	11.5%	5.7%	4.1%	2.0%	1.4%

implementation does omit some low-level details; for example, it does not recover gracefully from host timeouts/disconnects, and it is not hardened against timing-analysis attacks. We do not expect these omitted details to significantly affect the performance characteristics of our implementation. We measure the performance of each protocol piece in isolation (Sections 7.2–7.3) and also provide holistic end-to-end estimates of Stormy's performance (Section 7.4).

7.1 Methodology

We use the Shadow network simulator [41] to analyze the protocols' network costs. Shadow is a tool frequently used to simulate the Tor network and modifications/additions to Tor's protocols [40, 46, 64]. Using Shadow allows us to run our protocol implementations at network-scale and on networks with properties that accurately model the Tor network. We explore ranges of network parameters in our experiments, e.g., by varying bandwidth, latency, and committee size. We measure the time required to complete an honest protocol execution and the number of application-layer bytes transmitted by each host. Usually in our protocols the hosts send and receive an equal number of bytes; in cases of asymmetry we present "data transmitted (TX)" as the max of bytes sent and received.

Table 3 shows the default network parameters used across Shadow experiments. In the Authority Model, a single Computation Committee with high bandwidth (1 Gbps by default) performs both the offline preprocessing and the online computation. We use a one-way latency of 50 ms between the authorities, which corresponds to the median latency between Tor relays reported in the 2015 measurement study of Cangialosi et al. [16]. In the Relay Model, many parallel Triple Committees generate and transfer authenticated triples; the Computation Committee generates authenticated bits, receives relay inputs, and performs the online computation. For the default bandwidth allocations of the TC and CC members, we used the median active-committee bandwidths of 100 simulations of the committee-election process on a 2018-10-01 Tor consensus with $m_{\mathsf{TC}} = 1,000$ sampled TCs, $m_{\mathsf{CC}} = 8$ sampled CCs, and a committee size of $c = 25$. We conservatively set the committee members' latencies to 250 ms, which corresponds to the highest latency measured between any two Tor relays [16]. In both RelMode and AuthMode, when running input-sharing experiments, we use 7,000 input parties, which is an upper bound on the number of Tor relays in 2018-10. All input parties are configured pessimistically with 20 kbps links, which is the minimum bandwidth advertised by any relay in the Tor network.

Using Shadow comes with a couple of limitations. First, Shadow measures only the *network performance* of the protocols; computational performance (e.g. CPU usage or memory consumption) is not captured in the simulations. However, the protocols used in Stormy are computationally inexpensive. The offline preprocessing consists primarily of symmetric-key operations, and the online computation requires almost no cryptographic operations. Communication costs dominate protocol runtime, and so our Shadow experiments should closely estimate total protocol runtime. Second, it takes prohibitively long to simulate network-scale operation on long timescales (e.g. many hours). Therefore, when estimating the end-to-end performance in RelMode, we instead use a custom event simulator that incorporates results from the Shadow experiments (see Sec. 7.4).

Table 3: Default network parameters used in experiments.

	BW	Latency	Parties
RelMode			
Triple Committee	6.4 Mbps	250 ms	25
Computation Committee	12 Mbps	250 ms	25
AuthMode (CC)	1 Gbps	50 ms	5
Input Party	20 kbps	250 ms	7,000

7.2 Offline Preprocessing

7.2.1 Authenticated Triple Generation. Recall from Section 4 that one secret-shared authenticated AND triple is required for each AND gate evaluated during the online phase. Triple generation generally requires the most time and communication of all the components of Stormy. We evaluate the cost of triple generation by running experiments in which a single committee generates a batch of authenticated triples. The TC in RelMode generates batches of 5,112 triples at a time, and the CC in AuthMode generates batches of 280,000 triples. Smaller batch sizes require less memory, but reduce the protocol's efficiency. We choose batch sizes that are minimal at a given level of efficiency.

In the default configuration, a single TC in RelMode generates a batch of 5,112 triples in 210 s. 91 MiB of data is sent/received by each of the relays. The CC in AuthMode generates a batch of 280,000 triples in 9.0 s. 420 MiB of data is sent/received by each authority.

Figures 5.1–5.5 present the throughput and costs of generating triples when produced in various experimental setups. The AuthMode CC produces batches of 280,000 and a RelMode TC produces batches of 5,112 in all experiments. A single RelMode TC achieves modest throughputs in the range $[2, 43]$ $\frac{\text{triples}}{\text{second}}$ for realistic committee bandwidth capacities; however, since many TCs generate triples in parallel, RelMode's network-wide, overall throughput is much higher (7.4). The single AuthMode CC achieves much higher throughputs in the range $[2.7\text{k}, 36\text{k}]$ $\frac{\text{triples}}{\text{second}}$ because of their assumed high-bandwidth links. Each triple costs 18 KiB of communication per-party in the 25-member TC. In the 5-member CC, each triple costs only 1.5 KiB due to the smaller committee size and lower β-overhead achieved at the larger batch size (see Section 4).

Oblivious-transfer (OT) extensions performed pairwise between each host constitute the dominant cost of triple generation. We find that increasing available bandwidth can significantly improve runtime because the large messages sent during OT extension can be transmitted more quickly. Because each party performs 6 OT extensions with every other party, the runtime/transmission-cost of triple generation scales linearly with the size of the committee. The offline-phase protocols do not require many rounds of communication so latency has a minimal effect on runtime.

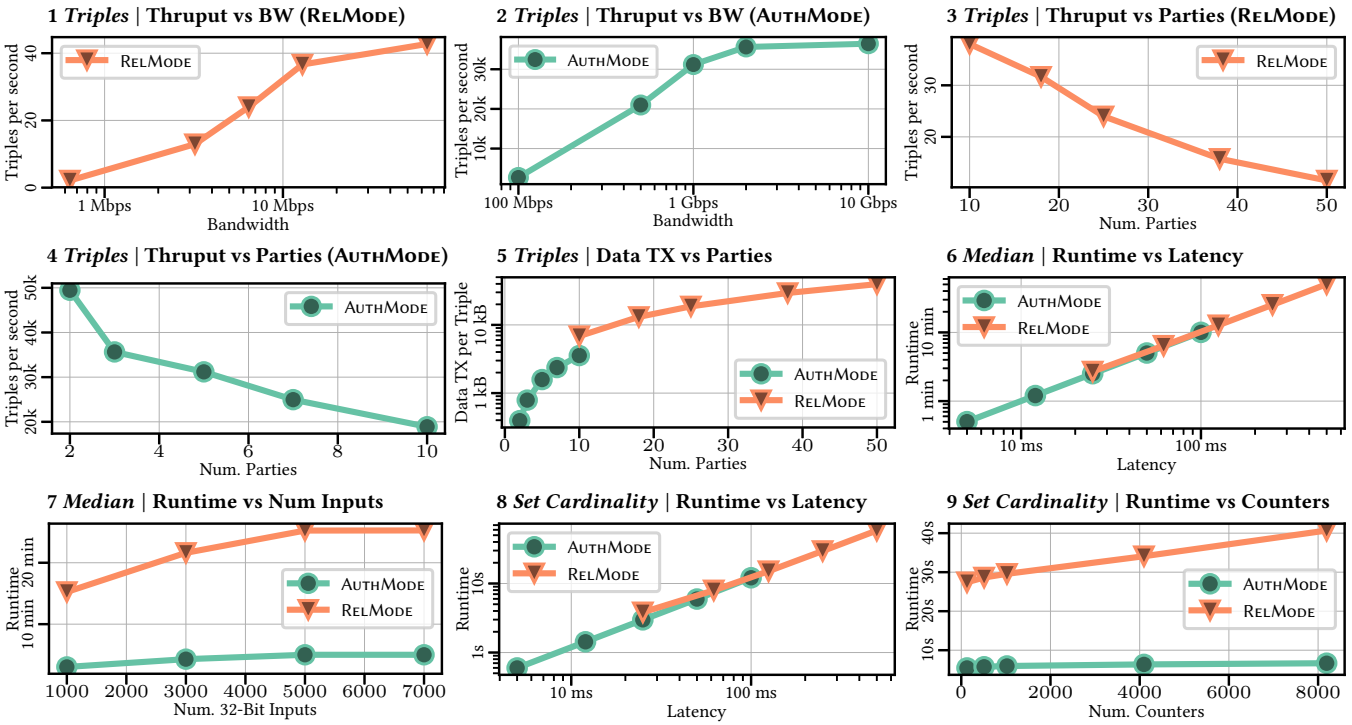

Figure 5: Network performance measurements of our implementation of the offline phase triple generation [68] and online phase evaluations of the median and set-union cardinality circuits (Section 6).

7.2.2 Authenticated Bit Generation. Authenticated bit generation, the other component of offline preprocessing required for input sharing, is orders-of-magnitude less expensive than triple generation. For brevity, we report results only in the default experimental configurations. 224,000 bits must be generated by the CC to perform the input sharing for a median computation with 7,000 32-bit inputs. 1,802,240 bits must be generated for a set-union cardinality computation with 1,024 counters, 32 entries per counter, and 55-bit entries. The RelMode CC generates 224k bits in 1.9 min using 73 MiB of communication; 1.8M bits takes 14 min and 579 MiB of communication. The AuthMode CC generates 224k bits in 2.8 s using 8.6 MiB of communication; 1.8M bits takes 3.8 s and 69 MiB of communication.

7.3 Online Computations

7.3.1 Input Sharing. Similar to bit generation, input sharing is the least expensive part of the online phase, so we present results only in the default configuration and only for the median circuit (recall from Section 6 that the online input-sharing protocol is not necessary for set-cardinality). With 7,000 input parties, RelMode takes 140 s and each relay in the CC transmits 89 MiB; AuthMode takes 24 s/16 MiB. We estimate that secret-sharing inputs for set-cardinality (in the default configuration) takes 92 s/60 MiB per CC member in RelMode and 92 s/320 MiB in AuthMode.

7.3.2 Median Circuit. For the median and set-union cardinality simulations, we assume the CC has already has the triples, bits, and input shares necessary for circuit evaluation. The combination of circuit AND depth and network latency almost completely determines online computation runtime; this is because the hosts must partially-open some authenticated bits after each AND gate level by sending and receiving bits to/from a designated party which incurs two one-way latency costs. This cost adds up quickly in our networks with relatively high latency. In contrast, communication costs are low as only a few bits are required per AND gate in the online-phase. Therefore, we do not show plots varying bandwidth and committee size as varying these parameters has minimal effect on runtime. We will observe that the online-phase runtime can be accurately predicted as $d \cdot \text{RTT}$, where d is the AND depth and the round-trip time (RTT) is twice the one-way latency.

When the number of inputs is not being varied, we set the median circuit to accept 7,000, 32-bit integer inputs. The CC in the RelMode computes this median in 25 min (the circuit has ~3,000 AND depth) using 10 MiB of communication. The AuthMode CC can compute this median in 5 min (7.1 MiB) due to their lower latencies.

Figures 5.6–5.7 plot the experimental runtime as latency and the number of 32-bit inputs is varied. As expected, runtime and latency share a linear relationship. Recall from Section 6 that our median circuit construction on n inputs of b bits each has AND depth of about $b \log_2^2(n)/2$. Accordingly, we find that the shape of the runtime-input graph is sublineary, which enables efficient scaling of the online median computation as the Tor network grows.

7.3.3 Set-Union Cardinality. In these experiments, by default, we use a set-union cardinality circuit with 1,024 counters, 32 entries per counter, and 55-bit counter entries. In this default configuration, the CC (in both models) can compute a cardinality estimate in under 30 *seconds*. This circuit requires less than 5 MiB of communication per CC member. The low runtime can be attributed to the shallow AND depth (= 21) of the circuit. As the Figure 5.9 runtime-counter plot shows, increasing the number of counters used in the set-union

cardinality circuit (and thus increasing the accuracy of the count) does not have a significant impact on the online protocol's runtime. However, keeping this number low keeps the number of triples required low. For example, increasing the number of counters from 1,024 to 8,096 only decreases the cardinality computation's standard error from 4% to 1% but yields a tenfold increase in the number of AND gates (and thus triples) required to compute the circuit.

7.4 System Performance

Here we use the preceding experimental results to estimate the performance of Stormy when computing the median and set cardinality, including both total time and communication. This estimation is straightforward for the Authority Model, as the authorities execute all parts of the system, and so we simply add up the time and communication of each piece (i.e. triple and bit generation for the offline phase, input sharing and computation for the online phase).

Performance estimation in the Relay Model is more complex, as it uses a large number of relays comprising sets of committees for the pieces of the protocol. The main challenge is analyzing the offline phase, which involves many differing relay bandwidths, relay churn, and performance over multiple computations. To perform this analysis, we build a custom simulator that models the execution of the offline phase over a single measurement epoch. Each simulation uses a sequence of published Tor consensuses [5] to determine the available relay population in each hour of the 24-hour epoch. The simulation process is then: (1) a set of Computation Committees (CCs) is sampled from the initial consensus and one chosen to be initially active, (2) a set of Triple Committees (TCs) is similarly sampled and a subset initially activated that has maximal throughput without violating memory or bandwidth constraints, (3) each active TC alternates between generating and transferring triples with the generation time determined by the TC's bandwidth and a linear regression on the experimental results over varying bandwidths, (4) TCs are queued to transfer triples to the CC with each transfer proceeding as fast as the two committees' bandwidths allow, (5) any time the CC is not receiving triples it generates batches of random bits at a speed based on its bandwidth and a linear regression of the experimental results, and (6) any committee with a member missing from an hourly Tor consensus dies at that hour, and some remaining sampled but inactive committees are activated in its place.

We simulate committee election on the Tor network for the day of 2018-10-01 using data from Tor Metrics [5]. As described in Section 2, the Tor network on that day consisted of 6,331 relays providing 275 Gbps in aggregate. Of these relays, we restrict ourselves to using the 5,506 that have the Fast and Running flags, as well as an archived descriptor, all of which Tor clients also require. Of these, 2,381 were guards. We use the consensus weights to select relays and report bandwidth using the bandwidths advertised by the relays in their descriptors. Each relay is assumed to have a 250 ms one-way latency to each other relay.

We perform the simulation 100 times where the relays are generating offline material for the median circuit and 100 times where the relays are generating offline material for the set-union cardinality circuit. For the median simulations, we sample $m_{\mathsf{TC}} = 1,000$ TCs and $m_{\mathsf{CC}} = 8$ CCs. For the cardinality simulations, we sample $m_{\mathsf{TC}} = 375$ TCs and $m_{\mathsf{CC}} = 633$ CCs. Both yield 1,008 total sampled committees, but computing the median requires more triples and thus benefits from more TCs, while the computing the cardinality requires more bits and thus benefits from a higher-bandwidth CC. The randomness we vary between simulations is the random bit string used for committee election.

7.4.1 Committee and Churn Analysis. Here we provide results detailing: (1) the number of committees activated, (2) the bandwidth of each committee (i.e. the bandwidth allocated by all members to that committee), and (3) churn statistics across the committees. We limit this set of results to the median-circuit simulations because these simulations require the most multiplication triples and thus require the highest level of parallel preprocessing. Any value that can change over the course of a simulation is reported on a time-averaged basis, that is, averaged over all hours during the simulation. For statistics that we measure, we record the median value and interquartile range (IQR) across simulations.

During the simulations, we never observed that all the sampled Computation Committees failed. Indeed, 75% of the time, the first activated CC lasted the entire 24 hours. The maximum number of CCs that died at any point was 3, where we sampled $m_{\mathsf{CC}} = 8$ CCs in total. The median bandwidth of the CC was 11.0 Mbps with an IQR of [10.0, 13.1]. This bandwidth limits the speed of bit generation and triple transfer, which is why the maximum-bandwidth committee among those sampled and alive is used as the active one.

We sampled $m_{\mathsf{TC}} = 1,000$ TCs, but only 327 were active in the median case (IQR: [319, 333]). 712 of the 1,000 TCs (IQR: [703, 722]) died during the median 24-hour simulation. This resulted in a median bandwidth fraction usage of 21.2% (IQR: [20.8, 21.6]) over time over all active committees, even though 25% of the bandwidth was always used at the beginning of the simulation. Of the 5,506 active relays at the start of the simulation, a median of 2,473 (IQR: [2,446, 2,491]) were in at least one active committee. The median average TC bandwidth was 7.1 Mbps (IQR: [6.8, 7.4]). This is lower than the 11.0 Mbps CC bandwidth primarily due to the fact that all TCs were used and not just the highest-bandwidth one.

7.4.2 Overall System Performance. Overall performance estimates of the system when computing a single median or a single set-union cardinality are reported in Table 4. We compute these estimates using the preceding Shadow experiments and RelMode simulations. In all of our simulations, we record the communication required for each party and the time taken to complete each phase of the protocol. The **Time** column reports the measured time required to complete a phase, and the **Data TX** column reports the measured amount of data the each member of a TC or CC transmits during the phase. Note that, during triple generation in RelMode, because relays are sampled to serve on a number of TCs proportional to their bandwidth, relays who can provide more bandwidth will transmit more data. Therefore the **Data TX** value reported in the RelMode TC is computed for a relay with a 1 Gbps link (a relay with a 500 Mbps link, for example, would transmit one-half as much data). All other values in the table are not dependent upon the bandwidth allocations of the hosts running the protocol.

For median computations, the online communication is relatively low, and the time needed for the online phase is due to high round complexity and high assumed latency. Therefore, using pipelining (i.e. running the online phase concurrently with a subsequent offline

Table 4: Summary of Section 7 performance evaluation. Communication costs are reported per-party and reported broken down by committee type (TC or CC).

Phase	Median			Set Cardinality		
	Time	Data TX		Time	Data TX	
		TC	CC		TC	CC
AuthMode						
Offline	9.5 min	✗	26 GiB	1.1 min	✗	3.0 GiB
Online	5.3 min	✗	21 MiB	1.6 min	✗	302 MiB
RelMode						
Offline	40 min	28 GiB	430 MiB	8.7 min	2.9 GiB	700 MiB
Online	28 min	✗	99 MiB	2.0 min	✗	61 MiB

phase), we estimate that 151 medians can be computed every 24 hours in AuthMode and 36 in RelMode. For set-union cardinality, input sharing becomes expensive due to the larger relay inputs, but even without pipelining, we estimate 533 daily computations can be performed in AuthMode and 134 in RelMode.

8 RELATED WORK

8.1 Tor Measurement

Previous work on privacy-preserving Tor measurement has focused on applying partially-homomorphic cryptosystems to the problem, limiting their functionality and security. The PrivEx [30] and PrivCount [42] systems can only provide simple sums of the relay inputs. Moreover, a single malicious relay can add in an arbitrary error term to the result, for example adding a random value to its input and making the sum useless. HisTorε [53] is designed to solve this problem by allowing each relay a limited number of input bits. However, it requires an analyst that cannot collude with any of the aggregating parties. Melis et al. [56] point out the usefulness of the median to robustly aggregate Tor inputs and suggest computing it using a count sketch. Their protocol reveals significantly more information about the inputs, however, as it performs a binary search on the space of possible input values that reveals the number of inputs in the search interval at every step. It is also vulnerable to an input party that doesn't prepare its sketches properly, where handling such a case is the main goal of computing the median. The PeerFlow bandwidth-measurement system [45] requires a robust estimate of the relays' bandwidth, and it presents a method to securely compute a median by securely computing tallies for bins covering the space of input values. This is both approximate and reveals more about the inputs than just their median. PSC [31] securely computes an aggregate unique count, but it uses a hash-table representation that grows linearly with the maximum count and is thus exponentially more expensive than our proposed method. The preceding systems generally allow the relays to store and update measurements obliviously and provide differentially-private outputs [28]. Their techniques for oblivious storage are compatible with and orthogonal to our system for aggregation. Our MPC protocols can compute differentially-private outputs, but we leave the design of such functions to future work. The Prio system [21] describes how inputs can be securely tested for validity as expressed by an abritrary Boolean circuit. Similar to our system, the Prio protocol uses offline/online MPC protocols, although the offline material can be supplied by the input party for efficiency. This technique can complement our system by providing efficient input validation, but it does not replace the need for robust statistics, as a valid input may still be a relative outlier.

8.2 MPC (Multi-party Computation)

Secure computation was originally introduced by several seminal works in the 1980s [11, 33, 70]. More recently, a long line of work starting with the work of Damgård et al. [23] has focused on building efficient secure computation in the preprocessing model (e.g., [15, 22, 32, 49]). These protocols introduced the notion of computing on *authenticated* values, and we follow this approach in our work. To the best of our knowledge, the largest experiment for real-world deployment of MPC was given by Wang et al. [68] showing global-scale MPC between 128 parties.

Another related line of work uses committee election to reduce communication required in large-scale MPC protocols. Original protocols in this area (e.g., [14, 19, 24, 25, 63, 72]) have focused on selecting committees that are poly-logarithmic in n, ensuring that each committee retains an honest majority. For $n = 7,000$, these committee sizes are prohibitively large for the $O(n^2)$ communication of malicious majority MPC protocols, such as the one we use in our work. It is an interesting question whether we could use honest majority committees in a practical solution.

A more similar approach to our own is taken by Choudhury et al. [20] who focus on reducing the communication of MPC for evaluating large circuits. They also use small committees with at least one honest party. Their main focus is to ensure robustness in case of abort, for which they only use one committee at a time. While they point out that computations can be performed in parallel, they do not focus on maximizing throughput, as we do here.

Hazay et al. [35, 36] propose the TinyKeys protocol for bandwidth-efficient triple generation. TinyKeys uses two committees: one (P_1) with at least one honest party, and the other (P_h) with many honest parties, where increasing the number of honest parties h in P_h reduces the key length ℓ and thus the communication. In the setting of randomly-sampled committees from a network that is at most 25% malicious, over the set of values for h and ℓ considered for actively-secure TinyKeys (Table 2 [35]), TinyKeys achieves at most a 6.5x decrease in communication complexity compared to the protocol of Wang et al. [68]. In contrast, Stormy increases the triple-generation rate by improving bandwidth utilization via multiple parallel triple committees. In our simulations, Stormy uses up to 1,000 committees simultaneously to utilize as much as 25% of Tor's bandwidth, where even the highest-bandwidth committee in those simulations could only use 0.19% of the total bandwidth by itself. Therefore, we estimate that we increase by $25/0.19 \approx 132$x the bandwidth available for triple production compared to using a single committee. Communication costs dominate triple production, and so we estimate that we increase the triple-generation rate over TinyKeys by about $132/6.5 \approx 20.3$x (see the technical report [65] for details).

These approaches seem orthogonal to some extent. It may be possible to run parallel executions of TinyKeys, thus combining the reduced communication of TinyKeys with the increased bandwidth

utilization of Stormy. Challenges to making this work include (1) TinyKeys uses short MAC shares, but the MAC switching protocol (Figure 2) that we use for parallel composition requires MAC shares long enough to ensure unpredictable hashes; and (2) TinyKeys uses larger committees to ensure more honest parties, but a large committee is more vulnerable to churn, reducing the number of triple batches it can compute before dying.

9 CONCLUSION AND FUTURE WORK

Although Tor is an important tool for online privacy, experience has shown that completely hiding all information about the network harms its mission by making the network hard to defend, understand, and improve. We show that MPC can provide a high level of control for such decisions. Our implemented system could be used today to solve existing problems in Tor, and it opens up possibilities that could not previously be contemplated. Our work suggests several valuable directions for future work, including improving resilience by adapting MPC protocols with identifiable abort [38] to identify and then remove misbehaving parties, designing differentially-private statistics with low circuit complexity, and applying secure computation to proactively detect and mitigate attacks, failures, and errors in Tor.

ACKNOWLEDGEMENTS

This work has been partially supported by the Office of Naval Research, the Defense Advanced Research Project Agency (DARPA), and the Department of Homeland Security Science and Technology Directorate, Homeland Security Advanced Research Projects Agency, Cyber Security Division under agreement number FTCY1500057. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering or any other funding agency.

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited.

REFERENCES

[1] MIRACL. https://libraries.docs.miracl.com/. Accessed: 2019-07-02.
[2] OpenSSL. https://www.openssl.org/. Accessed: 2019-07-02.
[3] Sodium. https://legacy.gitbook.com/book/jedisct1/libsodium/details. Accessed: 2019-07-02.
[4] Tor directory protocol, version 3. https://gitweb.torproject.org/torspec.git/plain/dir-spec.txt. Accessed: 2019-07-02.
[5] Tor Metrics. https://metrics.torproject.org. Accessed: 2019-07-02.
[6] Tor path specification. https://gitweb.torproject.org/torspec.git/plain/path-spec.txt. Accessed: 2019-07-02.
[7] Tor shared random subsystem specification. https://gitweb.torproject.org/torspec.git/tree/srv-spec.txt. Accessed: 2019-07-02.
[8] Abdelberi, C., Manils, P., and Kâafar, M. A. Digging into Anonymous Traffic: A Deep Analysis of the Tor Anonymizing Network. In *Fourth International Conference on Network and System Security, NSS 2010* (2010).
[9] Bauer, K., McCoy, D., Grunwald, D., Kohno, T., and Sicker, D. Low-Resource Routing Attacks Against Tor. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007)* (2007).
[10] Beaver, D. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology - CRYPTO '91* (1991).
[11] Ben-Or, M., Goldwasser, S., and Wigderson, A. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (1988).
[12] Biryukov, A., Pustogarov, I., and Weinmann, R. Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization. In *2013 IEEE Symposium on Security and Privacy, SP 2013* (2013).
[13] Borisov, N., Danezis, G., Mittal, P., and Tabriz, P. Denial of Service or Denial of Security? In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007* (2007).
[14] Boyle, E., Goldwasser, S., and Tessaro, S. Communication Locality in Secure Multi-party Computation - How to Run Sublinear Algorithms in a Distributed Setting. In *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013* (2013).
[15] Burra, S. S., Larraia, E., Nielsen, J. B., Nordholt, P. S., Orlandi, C., Orsini, E., Scholl, P., and Smart, N. P. High Performance Multi-Party Computation for Binary Circuits Based on Oblivious Transfer. Cryptology ePrint Archive, Report 2015/472, 2015.
[16] Cangialosi, F., Levin, D., and Spring, N. Ting: Measuring and Exploiting Latencies Between All Tor Nodes. In *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015* (2015).
[17] Chakravarty, S., Barbera, M. V., Portokalidis, G., Polychronakis, M., and Keromytis, A. D. On the Effectiveness of Traffic Analysis against Anonymity Networks Using Flow Records. In *Passive and Active Measurement - 15th International Conference, PAM 2014* (2014).
[18] Chou, T., and Orlandi, C. The Simplest Protocol for Oblivious Transfer. In *Progress in Cryptology - LATINCRYPT 2015* (2015).
[19] Choudhury, A. Breaking the $O(n|C|)$ Barrier for Unconditionally Secure Asynchronous Multiparty Computation - (Extended Abstract). In *Progress in Cryptology - INDOCRYPT 2013* (2013).
[20] Choudhury, A., Patra, A., and Smart, N. P. Reducing the Overhead of MPC over a Large Population. In *Security and Cryptography for Networks - 9th International Conference, SCN 2014* (2014).
[21] Corrigan-Gibbs, H., and Boneh, D. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017* (2017).
[22] Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., and Smart, N. P. Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits. In *Computer Security - ESORICS 2013* (2013).
[23] Damgård, I., Pastro, V., Smart, N. P., and Zakarias, S. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology - CRYPTO 2012* (2012).
[24] Dani, V., King, V., Movahedi, M., and Saia, J. Brief announcement: Breaking the $O(nm)$ Bit Barrier: Secure Multiparty Computation with a Static Adversary. In *ACM Symposium on Principles of Distributed Computing, PODC '12* (2012).
[25] Dani, V., King, V., Movahedi, M., and Saia, J. Quorums Quicken Queries: Efficient Asynchronous Secure Multiparty Computation. In *Distributed Computing and Networking - 15th International Conference, ICDCN 2014* (2014).
[26] Dingledine, R., Mathewson, N., and Syverson, P. F. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium* (2004).
[27] Durand, M., and Flajolet, P. Loglog Counting of Large Cardinalities (Extended Abstract). In *Algorithms - ESA 2003* (2003).
[28] Dwork, C., and Roth, A. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science 9*, 3-4 (2014).
[29] Elahi, T., Bauer, K. S., AlSabah, M., Dingledine, R., and Goldberg, I. Changing of the Guards: A Framework for Understanding and Improving Entry Guard Selection in Tor. In *Proceedings of the 11th annual ACM Workshop on Privacy in the Electronic Society, WPES 2012* (2012).
[30] Elahi, T., Danezis, G., and Goldberg, I. PrivEx: Private Collection of Traffic Statistics for Anonymous Communication Networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014).
[31] Fenske, E., Mani, A., Johnson, A., and Sherr, M. Distributed Measurement with Private Set-Union Cardinality. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017* (2017).
[32] Frederiksen, T. K., Keller, M., Orsini, E., and Scholl, P. A Unified Approach to MPC with Preprocessing Using OT. In *Advances in Cryptology - ASIACRYPT 2015* (2015).
[33] Goldreich, O., Micali, S., and Wigderson, A. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing* (1987).
[34] Goldwasser, S., and Lindell, Y. Secure Multi-Party Computation without Agreement. *J. Cryptology 18*, 3 (2005).
[35] Hazay, C., Orsini, E., Scholl, P., and Soria-Vazquez, E. Concretely Efficient Large-Scale MPC with Active Security (or, TinyKeys for TinyOT). In *Advances in Cryptology - ASIACRYPT 2018* (2018).
[36] Hazay, C., Orsini, E., Scholl, P., and Soria-Vazquez, E. TinyKeys: A New Approach to Efficient Multi-Party Computation. In *Advances in Cryptology - CRYPTO 2018* (2018).

[37] Heule, S., Nunkesser, M., and Hall, A. HyperLogLog in Practice: Algorithmic Engineering of a State of The Art Cardinality Estimation Algorithm. In *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings* (2013).

[38] Ishai, Y., Ostrovsky, R., and Zikas, V. Secure Multi-Party Computation with Identifiable Abort. In *Advances in Cryptology - CRYPTO 2014*.

[39] Jaggard, A. D., and Syverson, P. Onions in the Crosshairs: When The Man really *is* out to get you. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society* (2017).

[40] Jansen, R., Bauer, K. S., Hopper, N., and Dingledine, R. Methodically Modeling the Tor Network. In *5th Workshop on Cyber Security Experimentation and Test, CSET '12* (2012).

[41] Jansen, R., and Hopper, N. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012* (2012).

[42] Jansen, R., and Johnson, A. Safely Measuring Tor. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016).

[43] Jansen, R., Juárez, M., Galvez, R., Elahi, T., and Díaz, C. Inside Job: Applying Traffic Analysis to Measure Tor from Within. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018* (2018).

[44] Jansen, R., Tschorsch, F., Johnson, A., and Scheuermann, B. The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014* (2014).

[45] Johnson, A., Jansen, R., Hopper, N., Segal, A., and Syverson, P. PeerFlow: Secure Load Balancing in Tor. *Proceedings on Privacy Enhancing Technologies 2017*, 2 (2017).

[46] Johnson, A., Jansen, R., Jaggard, A. D., Feigenbaum, J., and Syverson, P. Avoiding The Man on the Wire: Improving Tor's Security with Trust-Aware Path Selection. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017* (2017).

[47] Johnson, A., Wacek, C., Jansen, R., Sherr, M., and Syverson, P. F. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13* (2013).

[48] Keller, M., Orsini, E., and Scholl, P. Actively Secure OT Extension with Optimal Overhead. In *Advances in Cryptology - CRYPTO 2015* (2015).

[49] Keller, M., Orsini, E., and Scholl, P. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016).

[50] Khattak, S., Fifield, D., Afroz, S., Javed, M., Sundaresan, S., McCoy, D., Paxson, V., and Murdoch, S. J. Do You See What I See? Differential Treatment of Anonymous Users. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016* (2016).

[51] Kissner, L., and Song, D. X. Privacy-Preserving Set Operations. In *Advances in Cryptology - CRYPTO 2005* (2005).

[52] Kolesnikov, V., Sadeghi, A., and Schneider, T. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In *Cryptology and Network Security, 8th International Conference, CANS 2009* (2009).

[53] Mani, A., and Sherr, M. HisTorϵ: Differentially Private and Robust Statistics Collection for Tor. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017* (2017).

[54] McCoy, D., Bauer, K. S., Grunwald, D., Kohno, T., and Sicker, D. C. Shining Light in Dark Places: Understanding the Tor Network. In *Privacy Enhancing Technologies, 8th International Symposium, PETS 2008* (2008).

[55] McLachlan, J., Tran, A., Hopper, N., and Kim, Y. Scalable Onion Routing with Torsk. In *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009* (2009).

[56] Melis, L., Danezis, G., and Cristofaro, E. D. Efficient Private Statistics with Succinct Sketches. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016* (2016).

[57] Miller, A., Xia, Y., Croman, K., Shi, E., and Song, D. The Honey Badger of BFT Protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016).

[58] Mittal, P., and Borisov, N. ShadowWalker: Peer-to-peer Anonymous Communication Using Redundant Structured Topologies. In *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009* (2009).

[59] Mittal, P., Wright, M. K., and Borisov, N. Pisces: Anonymous Communication Using Social Networks. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013* (2013).

[60] Nielsen, J. B., Schneider, T., and Trifiletti, R. Constant Round Maliciously Secure 2PC with Function-independent Preprocessing using LEGO. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017* (2017).

[61] Pinkas, B., Schneider, T., and Zohner, M. Faster Private Set Intersection Based on OT Extension. In *Proceedings of the 23rd USENIX Security Symposium* (2014).

[62] Rabin, T., and Ben-Or, M. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing* (1989).

[63] Saia, J., and Zamani, M. Recent Results in Scalable Multi-Party Computation. In *SOFSEM 2015: Theory and Practice of Computer Science - 41st International Conference on Current Trends in Theory and Practice of Computer Science* (2015).

[64] Sun, Y., Edmundson, A., Feamster, N., Chiang, M., and Mittal, P. Counter-RAPTOR: Safeguarding Tor Against Active Routing Attacks. In *2017 IEEE Symposium on Security and Privacy, SP 2017* (2017).

[65] Wails, R., Johnson, A., Starin, D., Yerukhimovich, A., and Gordon, S. D. Stormy: Statistics in Tor by Measuring Securely. Tech. rep., 2019.

[66] Wang, G., Luo, T., Goodrich, M. T., Du, W., and Zhu, Z. Bureaucratic Protocols for Secure Two-Party Sorting, Selection, and Permuting. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010* (2010).

[67] Wang, T., and Goldberg, I. Improved Website Fingerprinting on Tor. In *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013* (2013).

[68] Wang, X., Ranellucci, S., and Katz, J. Global-Scale Secure Multiparty Computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017* (2017).

[69] Winter, P., Köwer, R., Mulazzani, M., Huber, M., Schrittwieser, S., Lindskog, S., and Weippl, E. R. Spoiled Onions: Exposing Malicious Tor Exit Relays. In *Privacy Enhancing Technologies - 14th International Symposium, PETS 2014* (2014).

[70] Yao, A. C. How to Generate and Exchange Secrets (Extended Abstract). In *27th Annual Symposium on Foundations of Computer Science* (1986).

[71] Yin, M., Malkhi, D., Reiter, M. K., Golan-Gueta, G., and Abraham, I. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019* (2019).

[72] Zamani, M., Movahedi, M., and Saia, J. Millions of Millionaires: Multiparty Computation in Large Networks. Cryptology ePrint Archive, Report 2014/149, 2014.

A IDEAL FUNCTIONALITIES

This appendix contains the ideal functionalities used or realized by the protocols in the main body of the paper.

A.1 $\mathcal{F}_{\mathsf{Pre}}$

Functionality $\mathcal{F}_{\mathsf{Pre}}$, realized by Π_{Pre} (Figure 1) appears in Figure 6.

A.2 $\mathcal{F}_{\mathsf{Input}}$

Functionality $\mathcal{F}_{\mathsf{Input}}$, realized by Π_{Input} (Figure 4) appears in Figure 7.

A.3 $\mathcal{F}_{\mathsf{AccMsg}}$

The accountable message functionality $\mathcal{F}_{\mathsf{AccMsg}}$ used by Π_{Input} is given in Figure 8. This functionality is used to send messages between two parties in an accountable way. Specifically, it allows a sender (P_{in}) to send a (private) message m to a receiver R in the presence of a committee C in such a way that R cannot later deny that he received the message. This functionality consists of two procedures, send and reveal. The send procedure allows P_{in} to send a message to R in such a way that all parties in C agree whether a valid message was sent, without knowing the content of that message. The reveal procedure allows R to reveal the message m he received in send to the rest of C. Note that if send succeeds, then P_{in} will not be disqualified in the reveal procedure.

A.4 $\mathcal{F}_{\mathsf{Online}}$

We define an ideal functionality for the online phase of our protocol in Figure 9. The Computation Committee members provide the authenticated inputs from $\mathcal{F}_{\mathsf{Input}}$ and the MAC key and authenticated triples from $\mathcal{F}_{\mathsf{Pre}}$. The functionality reconstructs Δ and verifies that the inputs and triples are correctly authenticated under Δ. If not, it sends abort to all parties. Otherwise, it computes the circuit $\mathbb{C}$ given the inputs and returns the result to the adversary. The adversary then decides whether the functionality should return the output to the honest committee members.

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

Notation:

- Let CC be the Computation Committee.
- Let $\mathsf{TC}_1, \ldots, \mathsf{TC}_{m_{\mathsf{TC}}}$ be the Triple Committees.
- Let c be the size of each committee.
- Let $\mathcal{A}$ denote the set of parties controlled by the adversary
- Initialize $\Delta = \perp$.

Initialize: On input init from CC,

- Forward each input to $\mathcal{A}$ as it is received.
- If $\Delta = \perp$, set $\Delta \leftarrow \mathbb{F}_{2^\lambda}$, and randomly choose shares $\Delta^{(i)} \leftarrow \mathbb{F}_{2^\lambda}$ such that $\sum_{i=1}^{c} \Delta^{(i)} = \Delta$. Parties in $\mathcal{A} \cap \mathsf{CC}$ can choose their shares.
- If $\mathcal{A} \cap \mathsf{CC} \neq \emptyset$, allow $\mathcal{A}$ to input abort, which causes the functionality to abort and reject further calls.
- If $\mathcal{A}$ does not abort, output $\Delta^{(i)}$ to P_i.

Random: On input (random, $\mathbb{F}$, b) where $\mathbb{F} \in \{\mathbb{F}_{2^\lambda}, \mathbb{F}_2\}$ from each $P_i \in \mathsf{CC}$,

- Forward each input to $\mathcal{A}$ as it is received.
- For $h = 1, \ldots, b$, sample a random field element $r_h \leftarrow \mathbb{F}$.
- For $h = 1, \ldots, b$, produce random authenticated sharing $[\![r_h]\!]_{\mathsf{CC}}$ (i.e., compute $\mu_h = r_h \cdot \Delta$ and produce random sharings over CC of r_h and μ_h). Parties in $\mathcal{A} \cap \mathsf{CC}$ can choose their shares.
- If $\mathcal{A} \cap \mathsf{CC} \neq \emptyset$, allow $\mathcal{A}$ to input abort, which causes the functionality to abort and reject further calls.
- If $\mathcal{A}$ does not abort, for $h = 1, \ldots, b$, output $[\![r_h]\!]^{(i)}$ to $P_i \in \mathsf{CC} \backslash \mathcal{A}$.

Triples: On input (triples, ℓ) from each $P_i \in \mathsf{TC}_j$,

- Forward each input to $\mathcal{A}$ as it is received.
- For $h = 1, \ldots, \ell$, choose $x_h, y_h \leftarrow \mathbb{F}_2$, and set $z_h = x_h \cdot y_h$.
- For $h = 1, \ldots, \ell$, produce random authenticated sharings $[\![x_h]\!]_{\mathsf{CC}}$, $[\![y_h]\!]_{\mathsf{CC}}$, and $[\![z_h]\!]_{\mathsf{CC}}$ (i.e., for $w \in \{x_h, y_h, z_h\}$ compute $\mu = w \cdot \Delta$ and produce random sharings over CC of w and μ). Parties in $\mathcal{A} \cap \mathsf{CC}$ can choose their shares.
- If $\mathcal{A} \cap (\mathsf{TC}_j \cup \mathsf{CC}) \neq \emptyset$, allow $\mathcal{A}$ to input abort, in which case further calls from TC_j are rejected, abort is output to $P_i \in \mathsf{TC}_j$, and abort_j is output to $P_i \in \mathsf{CC}$. The functionality continues to respond to calls from other committees.
- If $\mathcal{A}$ does not output abort, for $h = 1, \ldots, \ell$, output $([\![x_h]\!]^{(i)}, [\![y_h]\!]^{(i)}, [\![z_h]\!]^{(i)})$ to $P_i \in \mathsf{CC} \backslash \mathcal{A}$.

Figure 6: Preprocessing functionality.

A.5 $\mathcal{F}_{\mathsf{RM-MPC}}$

We give the functionality for RelMode MPC in Figure 10. This is designed the setting in which a large number of parties wish to participate in a secure computation by providing inputs and (potentially) participating in the computation. To enable efficient computation, there is a single designated Computation Committee (CC) of size c. This committee performs the online phase of the computation and is allowed to abort the functionality. No other party can cause an abort, making this MPC functionality resilient to failure and malicious behavior by most parties.

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

Notation:

- Let P_{in} be the sender with input x.
- When a party P_i outputs (abort, P_{in}) this means he aborts and blames P_{in}. If he outputs (abort, C) this means he aborts and blames the committee.
- $S_{\mathsf{in}} \subseteq C$ is the set of parties blaming P_{in}, $S_C \subseteq C$ is the set of parties blaming the committee, and $S_{\mathsf{accept}} \subseteq C$ is the set of parties that blame nobody.

Authenticate Input: On input $\{x_h^i\}$ from P_{in}, where $h \in \{1, \ldots, b\}$, $i \in \{1, \ldots, c\}$,

- $S_{\mathsf{accept}} = C$.
- The functionality computes $x_h = \sum_{i=1}^{c} x_h^{(i)}$, and random shares of the authenticated value, $\{(\Delta x_h)^{(i)}\}$. It gives $\mathcal{A}$ the authenticated shares $\left(x_h^{(i)}, (\Delta x_h)^{(i)}\right)$ for $P_i \in \mathcal{A}$.
- Let $m_h^{(i)} = (\Delta x_h)^{(i)}$.
- If $P_{\mathsf{in}} \notin \mathcal{A}$ and $\mathcal{A} \neq \emptyset$, $\mathcal{A}$ partitions C into $(S_{\mathsf{accept}}, S_C)$.
- if $\mathcal{A} = \{P_{\mathsf{in}}\}$, $\mathcal{A}$ either sets $S_{\mathsf{in}} = C$, or $S_{\mathsf{accept}} = C$.
- If $\{P_{\mathsf{in}}\} \subset \mathcal{A}$, $\mathcal{A}$ partitions C into $(S_{\mathsf{accept}}, S_{\mathsf{in}}, S_C)$. Additionally, for each $P_i \in S_{\mathsf{accept}}$, $\mathcal{A}$ sets the values of $(x_h^{(i)}, m_h^{(i)})$, arbitrarily.
- The functionality sends (abort, C) to $P_i \in S_C$, (abort, P_{in}) to $P_i \in S_{\mathsf{in}}$, and $\left(x_h^{(i)}, m_h^{(i)}\right)$ to $P_i \in S_{\mathsf{accept}}$.

Figure 7: Input sharing functionality.

Functionality $\mathcal{F}_{\text{AccMsg}}$

Notation:

- Let P_{in} be the sender with input m.
- Let R be the receiver.
- Let C be the committee (Note that $R \in C$).
- When a party P_i outputs (abort, P_{in}) this means he aborts and blames P_{in}. If he outputs (abort, C) this means he aborts and blames the committee.

Send: On input (send, m) from P_{in},

- The functionality stores m.
- If $P_{\text{in}} \notin \mathcal{A}$, output (accept, m) to R, and output accept to all parties in $C \setminus \{R\}$.
- If $P_{\text{in}} \in \mathcal{A}$, do the following:
  - If $R \notin \mathcal{A}$, allow $\mathcal{A}$ to specify an input in {accept, abort}. If $\mathcal{A}$ inputs accept, then R outputs (accept, m) and all parties in $C \setminus \{R\}$ output accept. If $\mathcal{A}$ inputs abort, then all parties in C (including R) output (abort, P_{in}) (i.e., they abort and blame P_{in}).
  - If $R \in \mathcal{A}$, allow $\mathcal{A}$ to specify a partition of C, (S_{accept}, S_{in}, S_C). All $P_i \in S_{\text{accept}}$ output accept (if $R \in S_{\text{accept}}$, he additionally outputs m). All $P_i \in S_{\text{in}}$ output (abort, P_{in}). All $P_i \in S_C$ output (abort, C) (i.e., they abort and blame the committee).

Reveal: On input (reveal) from R,

- If $R \notin \mathcal{A}$, the functionality outputs (accept, m) to all parties in C.
- If $R \in \mathcal{A}$ and $P_{\text{in}} \notin \mathcal{A}$, $\mathcal{A}$ specifies a partition of C, (S_{accept}, S_C). Every $P_i \in S_{\text{accept}}$ outputs (accept, m). Every $P_i \in S_C$ outputs (abort, C).
- If both $R \in \mathcal{A}$ and $P_{\text{in}} \in \mathcal{A}$, $\mathcal{A}$ specifies a partition as above, and, additionally, for each party $P_i \in S_{\text{accept}}$, $\mathcal{A}$ specifies a message m_i'. Then, every player $P_i \in S_{\text{accept}}$ outputs (accept, m_i') and $P_i \in S_C$ output (abort, C).

Figure 8: Accountable messaging functionality.

Functionality $\mathcal{F}_{\text{Online}}$

Notation:

- The functionality is parametrized by a Boolean circuit $\mathbb{C}$.
- Let CC = $\{P_1, \ldots, P_c\}$ be the Computation Committee.
- Let $x_1, \ldots, x_n$ be all of the input bits successfully provided by all parties during the input sharing phase.
- Let ℓ denote the total number of AND gates in the circuit $\mathbb{C}$, and $(\llbracket w_j \rrbracket^{(i)}, \llbracket y_j \rrbracket^{(i)}, \llbracket z_j \rrbracket^{(i)})$ denote the ith party's shares of the jth triple.

Compute:

Party $P_i \in$ CC provides the following input: $\Delta^{(i)}$, $\{(\llbracket w_h \rrbracket^{(i)}, \llbracket y_h \rrbracket^{(i)}, \llbracket z_h \rrbracket^{(i)})\}_{h=1}^{\ell}$, $\{(\llbracket x_h \rrbracket^{(i)}, \llbracket \Delta x_h \rrbracket^{(i)})\}_{h=1}^{n}$

- If not every member of CC provides the same number of input shares and triple shares, output abort to every party.
- Reconstruct Δ from the shares provided.
- Reconstruct and verify the input values, $x_1, \ldots, x_n$ from the shares provided. If verification fails for any input value, output abort to every party.
- Reconstruct and verify the triples from the shares provided. If any triple is invalid ($z \neq w \cdot y$), or if verification fails for any triple value, output abort to every party.
- Compute $\mathbb{C}(x_1, \ldots, x_n)$ and output the result to $\mathcal{A}$.
- If $\mathcal{A}$ says continue, send $\mathbb{C}(x_1, \ldots, x_n)$ to the remainder of CC. Otherwise, send abort to the remainder of the CC.

Figure 9: Functionality for computing the online phase.

Functionality $\mathcal{F}_{\text{RM-MPC}}$

Notation:

- The functionality is parametrized by a Boolean circuit $\mathbb{C}$ outputting o bits.
- Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of all parties, and let CC = $\{P_{i_1}, \ldots, P_{i_c}\}$ be the Computation Committee.

Compute: On input $(x_1, \ldots, x_n)$ from $\mathcal{P}$ (each party supplies one input):

- If $\mathcal{A} \cap \text{CC} \neq \emptyset$, then $\mathcal{A}$ inputs either abort or run to the functionality. If $\mathcal{A}$ inputs abort, then the functionality outputs $\perp$ to all parties.
- For any $P_i \in \mathcal{A}$, $\mathcal{A}$ may input (abort, i) to the functionality (i.e., if an input party aborts), in which case the functionality sets $x_i = \perp$ and outputs (abort, i) to Computation Committee.
- The functionality computes $(y_1, \ldots, y_o) = \mathbb{C}(x_1, \ldots, x_n)$
- If $\mathcal{A} \cap \text{CC} \neq \emptyset$, then the functionality returns y to $\mathcal{A}$. $\mathcal{A}$ specifies a set $S_{\text{abort}} \subseteq \mathcal{P}$.
- The functionality outputs $\perp$ to all $P_i \in S_{\text{abort}}$ and outputs $(y_1, \ldots, y_o)$ to all $P_i \in \text{CC} \setminus S_{\text{abort}}$.

Figure 10: Large-Scale MPC functionality.