Distributed Measurement with Private Set-Union Cardinality

Ellis Fenske*
Tulane University

Akshaya Mani*
Georgetown University

Aaron Johnson
U.S. Naval Research Laboratory

Micah Sherr
Georgetown University

ABSTRACT

This paper introduces a cryptographic protocol for efficiently aggregating a count of unique items across a set of data parties privately — that is, without exposing any information other than the count. Our protocol allows for more secure and useful statistics gathering in privacy-preserving distributed systems such as anonymity networks; for example, it allows operators of anonymity networks such as Tor to securely answer the questions: *how many unique users are using the distributed service?* and *how many hidden services are being accessed?* We formally prove the correctness and security of our protocol in the Universal Composability framework against an active adversary that compromises all but one of the aggregation parties. We also show that the protocol provides security against adaptive corruption of the data parties, which prevents them from being victims of targeted compromise. To ensure *safe* measurements, we also show how the output can satisfy differential privacy.

We present a proof-of-concept implementation of the private set-union cardinality protocol (PSC) and use it to demonstrate that PSC operates with low computational overhead and reasonable bandwidth. In particular, for reasonable deployment sizes, the protocol run at timescales smaller than the typical measurement period would be and thus is suitable for distributed measurement.

CCS CONCEPTS

• **Security and privacy** → Distributed systems security;

KEYWORDS

secure computation; privacy-preserving measurement

1 INTRODUCTION

Gathering statistics is a critical component of understanding how distributed systems are used and/or misused. In privacy-preserving distributed systems, such as anonymity networks, the statistics-gathering process is complicated by the system's privacy requirements. Naïvely recording statistics poses significant risks to the system's users and the use of such techniques [37] has been widely debated [46]. Ideally, the statistics should be gathered privately — that is, nothing should be learned other than the statistic itself.

In the setting of anonymous communication, several recent papers have suggested methods of computing aggregate statistics at a quantifiable cost to anonymity. The PrivEx system of Elahi et al. [19] uses differential privacy [15] to privately collect statistics about the Tor network [14]. PrivCount [30], which we extend in this work, improves upon PrivEx by offering multi-phase iterative measurements while offering an optimal allocation of the ϵ privacy budget. Historϵ [36] is also inspired by PrivEx and uses histogram-based queries to provide integrity guarantees by bounding the influence of malicious data contributors.

While these secure measurement techniques significantly raise the bar for safe measurements in anonymity networks, they lack the ability to perform counts of *distinct* values among the data owners. For example, PrivEx, PrivCount, and Historϵ can answer the question—*how many clients were observed entering the Tor network?*—but cannot discern what fraction of the result constitutes unique clients. That is, they cannot answer the (perhaps) more useful question of *how many unique clients were observed on Tor?*

We refer to the problem of counting the number of unique values across data owners as *private set-union cardinality*. More formally, private set-union cardinality answers how many unique items are recorded across a set of Data Parties (DPs) while preserving the privacy of the inputs. Specifically, if there are d DPs and each DP_k contains a set of of zero or more items $\mathcal{I}_k$, we want to know $|\cup_{k=1}^{d} \mathcal{I}_k|$ without exposing any element of any item set to an active adversary. In instances where exact counts may reveal sensitive information, private set-union cardinality may naturally be combined with differential privacy to provide noisy answers.

Motivation. Private set-union cardinality is useful in many settings. For example, in the context of anonymity networks, it can determine how many *unique* users participate in the service, how many unique users connect via a particular version of client software, and how many unique destinations are contacted. Maintainers of anonymity networks can also use private set-union cardinality to determine how users regularly connect to the network (e.g., over mobile connections, through proxies, etc.) and how the network is used (e.g., the length of time that users spend on the network in a single session).

For structured overlay networks such as Chord [47], private set-union cardinality can determine the number of unique clients in the network and the number of unique lookups, without exposing users' identities.

For the web, private set-union cardinality can serve as a building block to determine the number of shared users among several sites, without revealing those users' identifiers.

*Co-first authors

CCS '17, October 30-November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134034

More generally, private set-union cardinality allows researchers and administrators of distributed systems to better understand how such systems are being accessed and used. It enables system designers to make informed, data-driven decisions about their systems based on actual usage safely when privacy is required.

A protocol for private set-union cardinality. The primary contribution of this paper is a private set-union cardinality protocol that we call PSC. We formally prove the correctness and security of PSC in the Universally Composable (UC) framework [6], ensuring that PSC is secure when (arbitrarily) composed with other instances of PSC and/or other UC protocols. PSC is robust against adaptive attacks against the DPs, so that if an adversary compromises a DP mid-collection, he learns only data collected post-compromise. The protocol is largely carried out by a set of Computation Parties (CPs) expected to be smaller than the set of DPs, and the protocol remains secure as long as at least one CP remains honest.

We additionally introduce an implementation of PSC, released as open-source software written in memory-safe Go. To achieve greater efficiency, our implementation uses some subprotocols that are not proven UC-secure, such as a verifiable shuffle [40] that is more practical than verifiable shuffles proven secure in the UC model (verifiable shuffles are explained in the next section). With these subprotocols, our implementation can still be proven secure in the classical (*i.e.* standalone) model and, as we demonstrate via at-scale evaluation experiments, incurs only moderate bandwidth and computation costs and can be practically deployed.

2 BACKGROUND

Before describing PSC, we briefly review some concepts and background that are necessary for understanding our algorithm.

Differential Privacy. Differential privacy [15] is a privacy definition that offers provable and quantifiable privacy of database queries. Differential privacy guarantees that the query responses look nearly the same regardless of whether or not the input of any one user is included in the database. Thus anything that is learned from the queries is learned independently of the inclusion of any single user's data [31]. Several mechanisms have been designed to provide differential privacy while maximizing accuracy [3, 16, 38, 42].

More formally, an (ϵ, δ)-differentially-private mechanism is an algorithm $\mathcal{M}$ such that, for all datasets D_1 and D_2 that differ only on the input of one user, and all $S \subseteq \text{Range}(\mathcal{M})$, the following holds:

$$Pr[\mathcal{M}(D_1) \in S] \leq e^{\epsilon} \times Pr[\mathcal{M}(D_2) \in S] + \delta. \quad (1)$$

ϵ and δ quantify the amount of privacy provided by the mechanism, where smaller values of each indicate more privacy. Dwork [15] proves that binomial noise, that is, the sum of n uniformly random binary values, provides (ϵ, δ)-differential privacy for queries that each user can affect by at most one when

$$n \geq \left(\frac{64 \ln(2/\delta)}{\epsilon^2} \right) \quad (2)$$

Eq. 2 presents a trade-off between privacy and utility, an issue inherent to differential privacy. Put alternatively, for the privacy level given by ϵ and δ, Eq. 2 yields the amount of binomial noise that is required to be added to the output of a query that each user can change by at most one. In this paper, we use this binomial noise technique to achieve differential privacy.

Discrete-log zero-knowledge proofs. PSC uses a few types of zero-knowledge proofs demonstrating knowledge of and relationships among the discrete logarithms of certain values. The values are elements in some group G of order q, and the discrete logs are with respect to some generator g (*e.g.* x is the discrete log of $y = g^x$, $x \in \mathbb{Z}_q$). In general, a zero-knowledge proof system [25] is a protocol between a prover $\mathcal{P}$ and verifier $\mathcal{V}$ in which the prover demonstrates the truth of some statement without revealing more than that truth, where the statement may be, for example, the existence or knowledge of a witness to membership in a language. Sigma protocols exist for the discrete-log statements PSC proves in zero-knowledge, where sigma protocols are three-phase interactive protocols starting with a commitment by the prover, followed by a random challenge from the verifier, and ended by a response from the prover. Such protocols can be made non-interactive via the the Fiat-Shamir heuristic [21], in which the random challenge is generated by the prover by applying a cryptographic hash to the commitment, and which is secure in the random-oracle model.

Verifiable shuffles. PSC makes use of *verifiable shuffles* [41]. Informally, a verifiable re-encryption shuffle takes as input ciphertexts, outputs a permutation of a re-encryption of those ciphertexts, and proves that the output is a re-encryption and permutation of the input. There are two security requirements for verifiable shuffles: privacy and verifiability. Privacy requires an honest shuffle to protect its secret permutation. Verifiability requires that any attempt by a malicious shuffle to produce an incorrect output must be detectable. Several protocols for verifiable shuffling have been proposed [2, 23, 27, 40]. As with the discrete-log proofs, the verifiable shuffles with interactive proofs can be made non-interactive using the Fiat-Shamir heuristic.

Secure broadcast. PSC uses a secure broadcast communication functionality. The security property that we require is broadcast with abort [26], which is slightly weaker than Byzantine agreement. This property guarantees that there exists some consensus output x such that each honest party that terminates either outputs x or aborts. Note that some honest parties may output x while others may abort. Broadcast with abort can be achieved given a PKI with the two-round echo-broadcast protocol. In this protocol, the sender sends a signed message to all the receivers, and each receiver append its own signature and sends it to all other receivers. If a party received the same message in every case and with valid signatures, it accepts that message, and otherwise it aborts.

3 OVERVIEW

PSC enables the secure computation of the cardinality of the union of itemsets that are distributed among a set of Data Parties (DPs). The protocol has two phases:

During the *data collection phase*, the DPs record observations in a vector of encrypted counters. These observations correspond to the metric of interest — for example, the unique clients observed by guard (entrance) relays in an anonymity network, or the unique exit points observed by the network's egress points. As explained below,

the counters are maintained in an oblivious manner, meaning that their plaintext cannot be revealed, even by the DPs that maintain them.

After the data has been collected, the *aggregation phase* proceeds as a series of cryptographic operations that, *in toto*, produces the cardinality of the union of the DPs' observations.

For clarity, the operation of these two phases is summarized below. The full details are found in Section 4.

Participants and threat model. The participants in the system are the d DPs and m Computation Parties (CPs). The latter are dedicated servers that apply verifiable shuffles and other cryptographic techniques to enable PSC.

Informally, our privacy guarantees are that (i) no information from an honest DP is ever exposed and (ii) the adversary cannot identify any individual data from the aggregated result (*i.e.* the cardinality). Formal security definitions and guarantees are provided in Section 5. Our protocol is secure as long as there is at least one honest CP. That is, we tolerate malicious DPs and CPs so long as at least one CP correctly obeys the protocol and the malicious CPs are selected statically. We prove our protocol secure in the UC framework, ensuring that PSC can be safely composed with other UC protocols.

Our protocol is not robust against dishonest DPs that report invalid statistics or dishonest CPs that choose to insert specific elements into the counted itemset. That is, like PrivEx [19] and PrivCount [30], PSC does not provide integrity guarantees that limit the influence of invalid data supplied by the malicious parties. We also note that a malicious CP can disrupt the protocol. We do not attempt to prevent malicious nodes from causing the protocol to abort.

Data collection phase and encrypted counters. Each DP maintains its dataset as a vector of encrypted counters. We assume a mapping between possible values in the DPs' itemsets and some finite range of integers. This mapping may be trivially realized by hashing the itemsets' values. More concretely, let $\mathcal{H}$ be a hash function that maps itemset values to integers in the range $[0, b-1]$. Conceptually, each DP stores a b-bit encrypted bit vector, where an encrypted 1 indicates that the DP has the corresponding item in its itemset.

An important property of our system — and one that is shared in existing work on privacy-preserving measurement systems [19, 30, 36] — is that the counters be stored *obliviously*. That is, after an initialization step, the DPs discard the keys used to encrypt the counters. They do, however, have the ability to transform any element in their encrypted counter to an (encrypted) 1, regardless of its previous (and unknowable) value. This construction allows the DPs to update the counters (e.g., to log the observation of a new client IP address) while maintaining resistance to *compulsion attacks*. That is, even under pressure to do so (*e.g.* in the form of a subpoena), DPs cannot release the plaintext of their encrypted counters.

Aggregation phase. During the aggregation phase, the DPs forward their encrypted counters to the CPs, which in turn perform a series of steps to securely compute the cardinality of the union.

In more detail, the aggregation phase proceeds as follows:

(1) **Counter forwarding:** Each DP creates secret shares of its encrypted counters and sends those shares to each of the CPs.
(2) **Noise addition:** Upon receiving the shares, the CPs collaboratively compute encrypted counters by homomorphically adding encrypted shares. The CPs then collaboratively generate n encrypted noise counters according to Eq. 2 and add the noise to their list of counters.
(3) **Counter shuffling:** The CPs next perform a collaborative verifiable shuffle, which mixes the noise and non-noise counters and obscures the mapping between a position in the vector and a particular itemset value.
(4) **Counter re-randomization:** To unlink the output counter values from the input values, each CP successively re-randomizes the values of the (shuffled) encrypted counters. This prevents a malicious DP from recognizing any output counter by "marking" it with a specific input value.
(5) **Counter decryption:** After the counters have been re-randomized, each CP removes its layer of encryption, eventually exposing the plaintext (but shuffled) counters.
(6) **Aggregation:** Finally, the CPs forward the shuffled counters to a single CP, which then computes the sum (*i.e.* counts the 1s in the vectors) and reports the result.

This process is summarized in Figure 1 and described more fully next.

4 PROTOCOL DETAILS

4.1 Preliminaries

PSC makes use of several cryptographic tools. A primary tool is ElGamal encryption [20], and PSC takes advantage of several capabilities of ElGamal, including distributed key generation and decryption, re-encryption, plaintext randomization, and multiplicative homomorphism. Let G be a Diffie-Hellman group for use by the ElGamal cryptosystem. Let q be the order of G, and let g be a generator of G. Let $x \xleftarrow{R} S$ denote a uniform random sample x from set S. In ElGamal, a keypair (x, y) is generated by choosing a private key $x \xleftarrow{R} \mathbb{Z}_q$ and setting the public key to $y = g^x$. A message $m \in G$, r is encrypted by choosing $r \xleftarrow{R} \mathbb{Z}_q$ and producing the ciphertext $(c_1, c_2) = (g^r, y^r m)$. A ciphertext (c_1, c_2) is decrypted as $m = c_2/c_1^x$. A ciphertext (c_1, c_2) is re-encrypted by choosing randomization parameter $r \xleftarrow{R} \mathbb{Z}_q$ and producing $(g^r c_1, y^r c_2)$, which is an encryption of the same plaintext. A ciphertext is re-randomized by choosing a randomization parameter $r \xleftarrow{R} \mathbb{Z}_q^*$ and producing (c_1^r, c_2^r), which, if the original encrypted value is $x \neq g^0$, makes the value uniformly random in $G \backslash \{g^0\}$, and otherwise doesn't change it.

We assume that PSC has access to two secure communication primitives as ideal functionalities (see details in Appendix A). The first is a secure point-to-point communication functionality $\mathcal{F}_{SC}$ that delivers messages from one party to another with confidentiality and authenticity. Such functionality can be realized using standard assumptions, even in the Universally Composable (UC) framework [6]. The second is a secure broadcast-with-abort functionality $\mathcal{F}_{BC}$ that guarantees that the same message is delivered to all honest parties that don't abort. Goldwasser and Lindell [26]

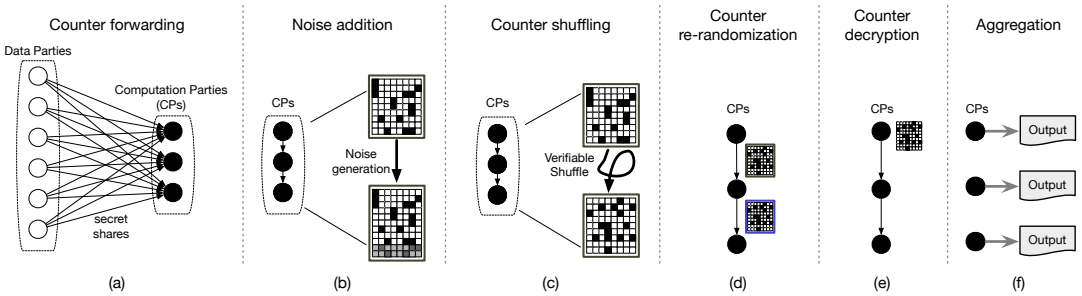

Figure 1: A simplification of the major steps of the private set-union cardinality protocol. Random shares of the encrypted counters are first forwarded from the DPs to the CPs (a). The CPs then add noise (b) and verifiably shuffle the counters (c). The encrypted counters are re-randomized to prevent linkability (d), and then decrypted (e) and finally aggregated (f) to yield the final output.

show how such a functionality can be implemented, including in the UC model. Details can be found in Appendix A.5.

We also provide PSC with the following functionalities for zero-knowledge proofs of knowledge, which output 1 if the inputs verify and output 0 otherwise (details in Appendix A):

(1) Discrete log ($\mathcal{F}_{ZKP\text{-}DL}$): For input (g, y) from the verifier, the prover inputs x such that $g^x = y$. This is defined in more detail in Appendix A.3.
(2) Discrete-log equality ($\mathcal{F}_{ZKP\text{-}DLE}$): For input (g_1, y_1, g_2, y_2) from the verifier, the prover inputs x such that $g_1^x = y_1 \wedge g_2^x = y_2$. This is defined in more detail in Appendix A.4.
(3) ElGamal re-encryption and re-randomization ($\mathcal{F}_{ZKP\text{-}RR}$): For input from the verifier of generator g, public key y, input ciphertext (c_1, c_2), and output ciphertext (d_1, d_2), the prover inputs re-encryption parameter r_1 and re-randomization parameter r_2 such that $((c_1 g^{r_1})^{r_2}, (c_2 y^{r_1})^{r_2}) = (d_1, d_2)$. This is defined in more detail in Appendix A.1.
(4) ElGamal shuffle ($\mathcal{F}_{ZKP\text{-}S}$): For an input from the verifier of shuffle inputs $((c_1^1, c_2^1), \ldots, (c_1^k, c_2^k))$ and shuffle outputs $((d_1^1, d_2^1), \ldots, (d_1^k, d_2^k))$, the prover inputs permutation π and re-encryption parameters $(r_1, \ldots, r_k)$ such that $(c_1^i g^{r_i}, c_2^i y^{r_i}) = \left(d_1^{\pi(i)}, d_2^{\pi(i)}\right)$. This is defined in more detail in Appendix A.2

The Schnorr proof [44] provides an efficient sigma protocol for $\mathcal{F}_{ZKP\text{-}DL}$, and the Chaum-Pedersen protocol [9] provides efficient sigma protocols for $\mathcal{F}_{ZKP\text{-}DLE}$ and $\mathcal{F}_{ZKP\text{-}RR}$. Hazay and Nissim [28] describe how to compile these (and sigma protocols in general) into efficient protocols secure in the UC model. Several zero-knowledge proofs of ElGamal shuffles of increasing efficiency have been presented [2, 23, 27, 40] (some require an honest verifier and must be strengthened for the malicious setting [24]). Wikström [51] shows how to obtain universally-composable proofs of correct shuffling.

4.2 Initialization and Data Collection

Each Data Party DP_i stores a hash table T^i with b bins. Each bin is a value in $\mathbb{Z}_q$. Let $\mathcal{H}$ be the hash function used and assume that it can be modeled as a random function. If we expect at most e inputs and desire that less than a fraction f of the inputs are expected to experience a collision, then we set $b = e/f$ to obtain the desired accuracy. To initialize each bin T_k^i, DP_i chooses values $r_k^{ij} \xleftarrow{R} \mathbb{Z}_q$, $1 \le j \le m$, sets $T_k^i = -\sum_j r_k^{ij}$, uses $\mathcal{F}_{SC}$ to send each r_k^{ij} to Computation Party CP_j, and then removes the r_k^{ij} from memory. We can view this process as secret sharing a stored value of $S_k^i = T_k^i + \sum_j r_k^{ij}$ among DP_i and the CPs, where S_k^i has an initial value of zero.

During data collection, a Data Party DP_i observes items and enters them into its hash table. To do so for a given item x, DP_i chooses $r \xleftarrow{R} \mathbb{Z}_q$ and updates bin $k = \mathcal{H}(x)$ as $T_k^i \leftarrow T_k^i + r$. This makes the secret-shared value S_k^i random. Such a value is non-zero with overwhelming probability (in q), and so, by interpreting a non-zero S_k^i to indicate an observed item, this process records the observation of item x, and S^i contains the set of such observations. We note that secret sharing the bins prevents the DP from storing any sensitive local state — an adversary that gains access to a DP's state would just see a uniformly and independently random value in each bin.

At the end of data collection, DP_i transfers its hash table to the CPs through additional secret sharing. That is, DP_i chooses values $s_k^{ij} \xleftarrow{R} \mathbb{Z}_q$, $1 \le j \le m-1$, $1 \le k \le b$, sets $s_k^{im} = T_k^i - \sum_{j=1}^{m-1} s_k^{ij}$, and sends each s_k^{ij} to CP_j through $\mathcal{F}_{SC}$.

4.3 Aggregating Inputs

The CPs begin the secure computation process by generating ElGamal encryption keys. CP_j chooses private key $x_j \xleftarrow{R} \mathbb{Z}_q$ and broadcasts public key $y_j = g^{x_j}$. This broadcast and all later ones are performed using $\mathcal{F}_{BC}$, and if this broadcast or any later one aborts, CP_j aborts. CP_j uses $\mathcal{F}_{ZKP\text{-}DL}$ as the prover with each CP_i as the verifier on y_j to prove knowledge of x_j. A group public key is computed as $y = \prod_j y_j$.

The CPs aggregate the hash tables by adding together each share they have received for a given bin. That is, CP_j computes an aggregate table A^j where the kth bin contains $A_k^j = \sum_{i=1}^{d} r_k^{ij} + s_k^{ij}$. Thus each A_k^j is a share of a value $A_k = \sum_j A_k^j$ that is random if any DP observed an item x such that $\mathcal{H}(x) = k$ and is zero otherwise. Therefore, under the interpretation that a non-zero value indicates the presence of an item in the table, A_k represents whether or not

some DP recorded an observation in bin k (except with negligible probability in q), and A contains the union of the sets S^i produced during data collection.

The CPs then prepare these aggregates for input into an ElGamal shuffle. CP_j encodes the kth value as $g^{A_k^j}$ and encrypts it using public key y to produce ciphertext $c_k^j = (g^r, y^r g^{A_k^j})$, $r \xleftarrow{R} \mathbb{Z}_q$. We place A_k^j in the exponent to turn the ElGamal multiplicative homomorphism into an additive operation on the A_k^j values, and no discrete-log operation is needed later to recover the desired plaintext values because we will only need to distinguish zero and non-zero exponent values.

CP_j then broadcasts c_k^j and computes encrypted shuffle inputs $c_k = \prod_j c_k^j$. Due to the ElGamal homomorphism, c_k is an encryption of g^{A_k}, and thus the values c_k represent an encryption of the union of sets observed at the DPs. CP_j uses $\mathcal{F}_{\text{ZKP-DL}}$ as the prover with each CP_i as the verifier on the first component of c_k^j (*i.e.* c_{k1}^j) to prove knowledge of r such that $g^r = c_{k1}^j$, which implies knowledge of the encrypted value $g^{A_k^j}$. If any proof fails to verify at CP_i (*i.e.* $\mathcal{F}_{\text{ZKP-DL}}$ outputs 0 to CP_i), then CP_i aborts. A verifier also aborts on the failed verification of any future proof, and so we will not explicitly state this again.

4.4 Noise Generation

The CPs collectively and securely generate noise inputs to provide differential privacy to the output. As discussed in Section 2, (ϵ, δ)-differential privacy can be provided to counting queries by sampling n bits uniformly at random, where n is the smallest value satisfying Inequality 2. The CPs generate such bits using verifiable ElGamal shuffles so that the resulting values are encrypted and can later be shuffled along with the encrypted inputs c_k.

The CPs run in parallel n ElGamal shuffle sequences, one for each noise bit. Each shuffle sequence has two input ciphertexts, $c_0^0 = (g^0, y^0 g^0)$ and $c_1^0 = (g^0, y^0 g^1)$, representing 0 and 1, respectively. Note that the encryption randomness is actually fixed at 0 so all CPs know that the inputs are correctly formed. Let $c^0 = \left(c_0^0, c_1^0\right)$. Then each CP_i, in sequence from $i = 1$ to m, performs the following actions:

(1) Re-encrypt c_0^{i-1} as c_0' and c_1^{i-1} as c_1'.
(2) Choose $\beta \xleftarrow{R} \{0, 1\}$, permute the re-encryptions as $c^i = (c_\beta', c_{1-\beta}')$, and broadcast c^i.
(3) Use $\mathcal{F}_{\text{ZKP-S}}$ as the prover with each other CP_j in parallel on shuffle input c^{i-1} and output c^i to prove that the shuffle was performed correctly.

From the final output c^m of the jth parallel noise shuffle we take the first element c_0^m to be the jth noise bit, which we denote c_{b+j}.

4.5 Shuffling, Re-randomization, and Decryption

At this point, we have produced $v = b+n$ values encoded as ElGamal ciphertexts c_k, where the first b contain the aggregated bins and the last n contain the noise. To hide the values of specific bins and noise bits, we have the CPs shuffle these values before decryption. Let $c^{1,0} = (c_1, \ldots, c_v)$. To shuffle $c^{1,0}$, each CP_i, in sequence from $i = 1$ to m, performs the following actions:

(1) Re-encrypt each ciphertext in $c^{1,i-1}$ to produce c'.
(2) Choose a random permutation π, permute c' as $c^{1,i} = \left(c'_{\pi(1)}, \ldots, c'_{\pi(v)}\right)$, and broadcast $c^{1,i}$.
(3) With each other CP_j in parallel, use $\mathcal{F}_{\text{ZKP-S}}$ as the prover on $c^{1,i-1}$ and $c^{1,i}$ to prove that the shuffle was performed correctly.

Next, we re-encrypt and then re-randomize the plaintexts of $c^{1,m}$. The re-randomization ensures that the encrypted values are each either g^0 or uniformly random in $G \backslash \{g^0\}$ and thus reveal only one bit of information . To accomplish this, let $c^{2,0} = c^{1,m}$, and then each CP_i, in sequence from $i = 1$ to m, performs the following actions:

(1) Re-encrypt each ciphertext in $c^{2,i-1}$ to produce c', re-randomize each ciphertext in c' to produce $c^{2,i}$, and broadcast $c^{2,i}$.
(2) With each other CP_j in parallel, use $\mathcal{F}_{\text{ZKP-RR}}$ as the prover on $c_k^{2,i-1}$ and $c_k^{2,i}$, for $1 \le k \le v$, to prove that the e-encryption and re-randomization was performed correctly. Each CP_j also verifies that, for all k, the ciphertext $(c_1, c_2) = c_k^{2,i}$ is such that $c_1 \neq g^0$ and aborts if not. This check ensures that the re-randomization parameter was non-zero.

Finally, we have the CPs decrypt the result. Let $c^{3,0} = c^{2,m}$. Each CP_i, in sequence from $i = 1$ to m, performs the following actions:

(1) Decrypt each ciphertext in $c^{3,i-1}$ using key x_i to produce $c^{3,i}$.
(2) For each $1 \le k \le v$, let $(c_1, c_2) = c_k^{3,i-1}$ and $(c_3, c_4) = c_k^{3,i}$, and, with each other CP_j in parallel, use $\mathcal{F}_{\text{ZKP-DLE}}$ as the prover on $(g, y_i, c_1, c_2/c_4)$. Each CP_j also verifies that $c_3 = c_1$ and aborts if not. These steps prove that the decryption was performed correctly.

Let p_i be the second component of $c_i^{3,m}$, which is a plaintext value, and let b_i be 0 if $p_i = g^0$ and be 1 otherwise. Each CP produces the output value $z = \sum_{i=1}^{v} b_i - n/2$, where the $-n/2$ term corrects for the expected amount of added noise.

4.6 Optimizations

We have given the PSC protocol in a way that is clear and amenable to proving security. There are a number of practical optimizations that can be made to it. These include the following:

- During initialization, instead of having the DPs send b random values to the CPs, the DPs can just send a short random seed, and then both sets of parties can expand it using a pseudorandom generator.
- Re-encryption, re-randomization, and decryption operations can be combined, which eliminates the added rounds and messages in the final sequential decryption. This requires a zero-knowledge proof for these combined operations.
- The zero-knowledge proofs can be made non-interactive via the Fiat-Shamir heuristic [21].
- The noise generation phase doesn't depend on any inputs from the DPs and thus can be done in advance, for example while the DPs are collecting data.

5 SECURITY ANALYSIS

PSC is designed to compute set union across the DPs while maintaining strong provable security and privacy properties. As we will show, PSC is secure against an active adversary that controls all but one of the CPs. Moreover, PSC provides *forward privacy*, that is, an adversary that corrupts a DP at some point during data collection learns nothing about what was observed in the past. Finally, the output is differentially private, which helps hide the existence of any specific item in the set union while providing an accurate estimate of the set-union size.

5.1 Protocol Security

We prove PSC secure in the Universally Composable framework [6]. This provides a strong notion of security and allows PSC to serve as a component of a larger system without losing its security properties. Our proof is in a "hybrid" model that assumes the existence of the functionalities used in the protocol description (see Section 4). Each functionality can be instantiated using any protocol that can be proven to UC-emulate the functionality. The PSC security proof in the UC model also directly implies its security in weaker models, such as the standalone model, and thus in order to improve efficiency in implementation we can accept a weaker security model and instantiate a functionality needed by PSC using a protocol that can only be proven in the weaker model (*e.g.* using the Neff shuffle for $\mathcal{F}_{ZKP\text{-}S}$). See Section 4.1 for a description of possible protocols to implement the functionalities used by PSC and their provable security properties.

An ideal functionality $\mathcal{F}_{PSC}$ for PSC is given in Figure 2. Note that, in addition to the CPs and DPs, the functionality interacts with an adversary A. We will show that PSC UC-emulates $\mathcal{F}_{PSC}$ against adaptive DP corruptions and static CP corruptions as long as one CP is honest. Security against adaptive corruptions is often difficult to prove with efficient protocols. However, we need adaptive security to express that nothing can be learned about past inputs by corrupting a DP, and so we allow adaptive corruptions against the DPs only.

Because PSC UC-emulates $\mathcal{F}_{PSC}$, it inherits the security properties that are satisfied by the functionality. $\mathcal{F}_{PSC}$ clearly produces the noisy set-union cardinality output, although it includes the limitations that the adversary can prevent any output at all and that the adversary is able to include arbitrary entries in the set even if he only controls a CP. The latter limitation is minimal, as the function to be computed already allows a corrupt DP to provide an input containing whatever the adversary wants. We can see that the functionality provides input privacy in that an adversary doesn't learn anything other than the inputs and outputs of the corrupted parties. We can also observe that the functionality provides forward privacy for the DPs, as a DP does not reveal anything about its past inputs upon corruption. Finally, the adversary does have the power to cause honest parties to abort at will, even potentially resulting in some honest parties aborting and others terminating successfully.

We prove that PSC UC-emulates $\mathcal{F}_{PSC}$ in a hybrid model using the following functionalities: $\mathcal{F}_{SC}$, $\mathcal{F}_{BC}$, $\mathcal{F}_{ZKP\text{-}DL}$, $\mathcal{F}_{ZKP\text{-}DLE}$, $\mathcal{F}_{ZKP\text{-}RR}$, $\mathcal{F}_{ZKP\text{-}S}$. See Section 4.1 for details on these functionalities. Let $\mathcal{M}$ be the hybrid model including these functionalities. The security claim for PSC is given in Theorem 5.1.

(1) Wait until (CORRUPT-CP, C) is received from A. For each $i \in C$, add CP_i to the list of corrupted parties.
(2) Upon receiving (CORRUPT-DP, i) from A, add DP_i to the list of corrupted parties. Note that past inputs to DP_i are not revealed to A upon corruption.
(3) Set $v^i \leftarrow 0^b$ for each DP_i. Upon receiving (INCREMENT, k) from DP_i, set $v^i_k \leftarrow 1$.
(4) Upon receiving GO from DP_i, ignore further messages from DP_i, and send (GO, DP_i) to A.
(5) Upon receiving GO from CP_i, ignore further messages from CP_i, and send (GO, CP_i) to A.
(6) Set $v^A \leftarrow 0^b$. Upon receiving (INCREMENT, k) from A, if any CP or DP is corrupted, set $v^A_k \leftarrow 1$.
(7) Sample the noise value as $N \sim \text{Bin}(n, 1/2)$, where $\text{Bin}(n, p)$ is the binomial distribution with n trials and success probability p.
(8) Let $w_i = \max\left(v^1_i, \ldots, v^d_i, v^A_i\right)$, $1 \le i \le b$. Compute output $z \leftarrow \sum_{i=1}^{b} w_i + N$. Once GO has been received from each CP, output z to CP_m.
(9) If at any point $\mathcal{F}_{PSC}$ receives (ABORT, IDS) from A, send ABORT to all parties indicated by IDS and do not send additional outputs besides other ABORT commands as directed by A.

Figure 2: $\mathcal{F}_{PSC}$: the ideal functionality for PSC

Theorem 5.1. *The PSC protocol UC-emulates $\mathcal{F}_{PSC}$ in the $\mathcal{M}$-hybrid model with respect to an adversary that adaptively corrupts the DPs and statically corrupts at most $m - 1$ CPs.*

Proof. We leave the proof to Appendix B.2. □

5.2 Protocol Privacy

PSC ensures that only the desired output is learned by an adversary, and it guarantees that the output doesn't itself violate privacy by making the output (ϵ, δ)-differentially private. The privacy notion is applied per-item; that is, the differential-privacy guarantee (see Equation 1) applies to two input sets that differ only in the existence of a single item. Thus the adversary cannot conclude whether or not a single item was observed by some DP. The extent to which the adversary can be made to suspect an item was observed is determined by the privacy parameters ϵ and δ. PSC can easily be configured to provide privacy for any small number k of items by reducing the ϵ and δ by a factor of k.

6 IMPLEMENTATION AND EVALUATION

We constructed an implementation of PSC in Go to verify our protocol's correctness and to measure the system's computation and communication overheads. We run experiments over large synthetic datasets and measure our implementation's performance. We describe the implementation details and design choices in Section 6.1 and then present our performance evaluation in Section 6.2.

6.1 Implementation

We built an implementation of PSC in 1427 lines of Go using the DeDiS Advanced Crypto Library for Go package[1]. ElGamal encryption is implemented in the NIST P-256 elliptic curve group [43] and the CPs use Neff's verifiable shuffle [40] to shuffle the ElGamal ciphertexts.

The CPs perform secure broadcast using the two-round echo-broadcast protocol (see Section 2). They use the Schnorr signature algorithm [44] over the NIST P-256 elliptic curve for signing and SHA-256 for computing digests. We rely on TLS 1.2 to provide secure point-to-point communication.

We use "Biffle" in the DeDiS Advanced Crypto Library for shuffling the noise vectors during the noise generation phase. Biffle is a fast binary shuffle for two ciphertexts based on generic zero-knowledge proofs.

For zero-knowledge proofs, we use Schnorr proofs [44] for knowledge of discrete log of the Elgamal public key and blinding factors, the Chaum-Pedersen proof [9] for equality of discrete log of decrypted Elgamal ciphertexts and public key, and the generalization of the Chaum-Pedersen proof [40] for the ElGamal ciphertext re-encryption and re-randomization. Non-interactive versions of all these proofs are produced using the Fiat-Shamir heuristic [21].

A single DP program emulates all the DPs in our implementation. However, in a real deployment, the DPs would be distributed.

To encourage the use of PSC by privacy researchers and practitioners, we are releasing PSC as free, open-source software, available for download at http://safecounting.com/.

6.2 Evaluation

Experiments are carried out on 8 to 32 core AMD Opteron machines with 32GB to 528GB of RAM running Linux kernel 3.10.0. Our implementation of PSC is currently single-threaded. Although the computational cost of PSC's noise generation is significant and may be done by the CPs before the inputs are received, we parallelize it so that it can be done on multicore machines after inputs are received . We use "parallel for" from the golang par package[2] for this parallel noise shuffling.

We instantiate all CPs and DPs on our 8 to 32 core servers. Google Protocol Buffers [50] is used for serializing messages, which are communicated over TLS connections between PSC's parties. We use Go's default crypto/tls package to implement TLS.

Query and dataset. We evaluate PSC by considering the query: what is the number of unique IP connections as observed by the nodes in an anonymity network? Rather than store 2^{32} (or, for IPv6, 2^{128}) counters, we assume b counters (where $b \ll 2^{32}$) and map IP addresses to a ($\lg b$)-bit digest by considering the first $\lg b$ bits of a hash function; this results in some loss of accuracy due to collisions. For each experiment, we chose an integer uniformly at random from the range $[0, 20000]$. Then for each DP, we choose from its counters a random subset of that size to set to 1; the remaining counters are set to 0.

We note that the performance of PSC is independent of the number of unique IPs. Therefore, in our performance evaluation, we are interested in how the number of counters b affects the operation of PSC, rather than the number of unique IPs.

Table 1: Default values for system parameters.

Parameter	Description	Default
b	number of counters	200,000
m	number of Computation Parties	5
d	number of Data Parties	20
ϵ	privacy parameter	0.3
δ	privacy parameter	10^{-12}

Experimental setup. The default values for the number of bins b, the number of CPs m, the number of DPs d, ϵ, and δ are listed in Table 1.

We determine these default values by considering which values would be appropriate for an anonymity network such as Tor. Currently, Tor reports 2 million user connections[3] and over 2,000 guard nodes[4]. Assuming that 1% of Tor guards deploy PSC, we have $d = 20$. Also, given this level of PSC deployment, we would expect a Tor guard to see approximately 20,000 unique IPs.

To measure the aggregate with high accuracy, we limit hash-table collisions to at most a fraction f of inputs in expectation by using a hash table of size $1/f$ times the number of inputs. Therefore, for $f = 10\%$, we set (unless otherwise specified) $b = 200,000$ in all our experiments. We set $\epsilon = 0.3$ as this is currently recommended for safe measurements in anonymity networks such as Tor [30]. To limit to 10^{-6} the chance of a privacy "failure" affecting any of 10^6 users, we set δ to 10^{-12} [17]. We set these default values as system-wide parameters, unless otherwise indicated.

Accuracy. The trade-off between accuracy and privacy is governed by the choice of ϵ and δ. We vary ϵ from 0.2 to 0.6, keeping the number of bins at $b = 200,000$. We found that values below 0.2 produced too much noise and offered low utility. Values of ϵ greater than 0.6 would not provide a reasonable level of privacy.

The actual and the noisy aggregate values along with the standard deviation for the noisy aggregates (the noise follows a binomial distribution) for different values of ϵ is shown in Table 2. We observe that the standard deviation of the noisy value is at most 106.44. Therefore, the noisy aggregates are very close to the actual aggregates, as expected. In summary, PSC gives highly accurate results for the desired privacy level.

Communication cost. PSC incurs communication overhead by transmitting symmetric-key-encrypted counters between the DPs and CPs and the ElGamal encrypted counters among the CPs. To be practical, a statistics gathering system should impose a low communication overhead for the DPs, which can have limited available bandwidth. However, we envision the CPs to be well-resourced dedicated servers that can sustain at least moderate communication costs.

We explore PSC's communication costs by varying the number of bins b, the number of CPs m, the number of DPs d, and ϵ. The

[1] http://github.com/dedis/crypto
[2] https://github.com/danieldk/par

[3] Tor uses simple statistical techniques to roughly estimate the number of users using the network. A major motivation of our work is to provide a more robust and accurate method of determining the size of Tor's user base.
[4] See https://metrics.torproject.org/.

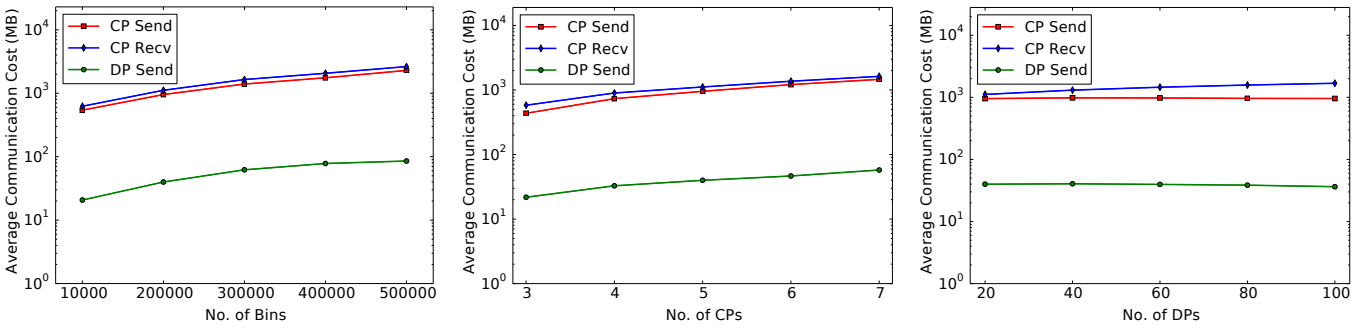

Figure 3: The communication cost incurred by the CPs and DPs varying the number of bins (*left*), the number of CPs (*center*) and the number of DPs (*right*).

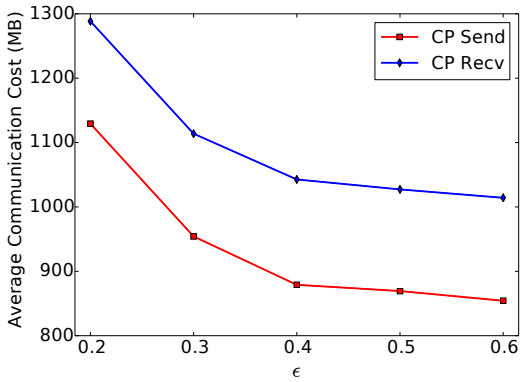

Figure 4: The communication cost incurred by the CPs varying ϵ.

Table 2: Actual, noisy aggregates, and standard deviation for various values of ϵ.

Epsilon	Actual Aggregate	Noisy Aggregate	Standard Deviation
0.2	4054	3989	106.44
0.3	4054	4028	70.96
0.4	4054	4051	53.22
0.5	4054	4027	42.58
0.6	4054	4129	35.48

average communication costs for the CPs and DPs are plotted in Figure 3 and Figure 4. We omit error bars as the variance in the communication cost incurred among the CPs was negligible; the same is true for the variance in communication cost among the DPs.

We first consider how the number of bins influences communication cost. We run PSC, varying b from 100,000 to 500,000, and plot the results in Figure 3. The values of the bins (*i.e.* `0` or `1`) do not affect the communication cost of the DPs, as a DP transmits an encrypted value for either `0` or `1`. For up to 200,000 bins, the communication cost for each DP is fairly modest. For example, if PSC is run once an hour, then the communication cost is approximately 60 MB/hr (16.67 KBps).

The communication costs are more significant for the CPs, which we envision are dedicated machines for PSC. With 200,000 bins, each CP requires a bandwidth of 954.20 MB for sending and 1.11 GB for receiving (approximately 300 KBps if executed once per hour).

We next consider how m, the number of CPs, affects the communication cost. We vary m from 3 to 7 and plot the results in Figure 3. The communication cost for the DPs increases at a much slower pace than that for the CPs. Even up to 7 CPs, the communication cost for each DP is fairly modest — approximately 60 MB (or 16.67 KBps, if run every hour).

The communication costs increase at a much faster pace for the CPs as each CP broadcasts proofs and ciphertexts to the other CPs. Therefore, for up to 5 CPs, each CP requires a modest bandwidth of 954.20 MB for sending and 1.11 GB for receiving (if run hourly, approximately 308 KBps).

We rerun PSC with different numbers of DPs. Figure 3 shows that varying the number of DPs has little effect on the average communication costs for the DPs and the CPs. This is because the number of encrypted counters transmitted by the DPs and the number of ElGamal encrypted counters transmitted by the CPs remain the same across these experiments.[5]

Lastly, we run PSC with different values of ϵ to determine how the choice of privacy parameter affects the communication costs. Figure 4 shows that the average communication costs for the CPs decrease when ϵ is increased. The communication costs for the CPs are reasonable even for a low value of $\epsilon = 0.2$. On average, each CP requires a bandwidth of at most 1.13 GB for sending and 1.29 GB for receiving (358 Kbps, if performed once an hour).

In summary, we find that PSC incurs reasonable communication overhead: the costs to DPs are modest, and, while slightly higher for CPs, they remain practical.

Overall runtime and computation cost. To understand the computation costs of PSC, we perform a series of microbenchmarks. We focus on evaluation of the CPs, as the DPs simply increment a counter whenever they observe a client connection. That is, the computation overhead of a DP is negligible.

[5]Currently, in our implementation, for all the above communication experiments, the DPs send the symmetric key share for each counter to every CP. We note that as a future optimization, this can be reduced to a great extent by using a Pseudo Random Function (PRF) and just sending a key per CP. The CPs can then use the PRF to generate the key-shares for every counter. We expect that this will reduce the communication cost of the DPs to half of the observed value.

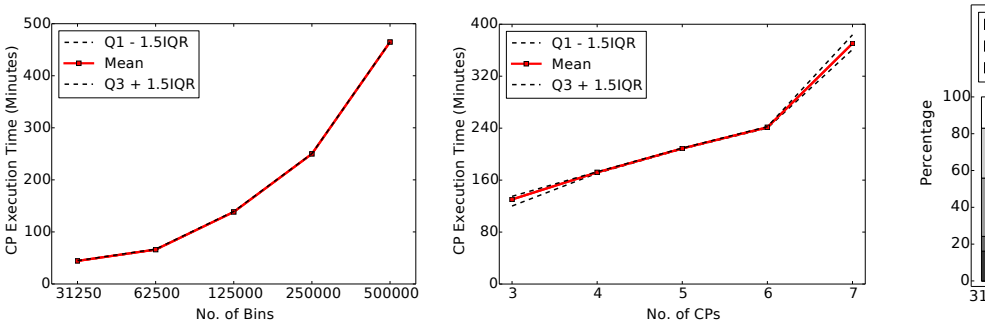

Figure 5: The average execution time as a function of the number of bins (*left*) and the number of CPs (*right*). The dashed-lines show the range within 1.5xIQR from the lower and upper quartiles.

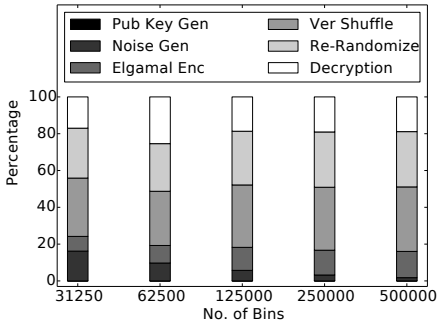

Figure 6: The computation cost incurred by the CPs for different operations, varying the number of bins.

We observe that the time taken for each operation of the CPs (including public key generation, parallel noise generation, ElGamal encryption, verifiable shuffling, re-randomization and re-encryption, and decryption) takes less than 3% of the total time taken in a run of the protocol. This is because the time required to transfer messages via secure broadcast overwhelms the time required for computation.

Figure 5 shows the overall running time (including the time required for network communication) as a function of the number of bins b and the number of CPs m. We first consider how number of bins b affects the computation cost (note that more data must be communicated via secure broadcast as b increases). The overall runtime is moderate. It takes approximately 1 hr 6 min for an experiment with 62,500 bins.

We next consider how the number of CPs m affects the computation cost. The computation cost increases with the CPs at a slower pace than with the bins. Even up to 6 CPs, the average computation cost for each CP is fairly modest — approximately 4 hr 9 min.

Still, to better understand the computational overhead of PSC, we measure the computation time as we vary the number of bins b. Figure 6 shows the distribution of processing overheads for the CPs for different operations, which account for less than 3% of the total execution time.

7 RELATED WORK

Privacy-preserving measurements. The operators of distributed systems have a critical need to understand how their systems are being used. The most straightforward measurement technique is to directly log and analyze the metric of interest. In the context of privacy-preserving systems, such as anonymity networks, this approach can be problematic, and attempts that directly measure user behavior in anonymity networks [37] have been heavily criticized. In particular, Soghoian [46] has called for improved community standards for research on anonymity networks. Loesing et al. [35] reiterate this need and propose privacy-preserving techniques for identifying trends in Tor. The techniques they describe for unique counting are to aggregate and round unique observations made at each relay. These techniques are not as safe as PSC, in that they only perform aggregate over time and not across relays, they require intermediate observations to be stored in memory, and they do not satisfy any rigorous privacy definition. Some of these techniques are also heuristic, as they indirectly observe client activity (*e.g.* directory downloads) and must guess how much activity corresponds to a unique user.

Recently, several measurement techniques have been proposed for Tor that use differential privacy to protect user privacy. Here, the goal is to produce useful aggregate statistics about the anonymity network, while providing quantifiable privacy guarantees. Elahi et al. [19] introduce secret-sharing and distributed-encryption variants of PrivEx, a system for private traffic statistics collection for anonymity networks. The PrivCount system, which serves as the inspiration of our work, extends PrivEx for collection of Tor statistics information, and introduces optimal allocation techniques for the ϵ privacy budget. Finally, Historyϵ [36], like PrivEx and PrivCount, use differential privacy to securely measure characteristics of the Tor network. Historyϵ adds integrity protections by bounding the influence of malicious data collectors (or, in our terminology, DPs). These systems can aggregate counts across nodes, but they cannot compute *unique* counts, which is the functionality provided by PSC.

Related cryptographic protocols. Brandt suggests a protocol [5] very similar to the aggregate-shuffle-rerandomize-decrypt scheme our CPs execute. However, our construction differs in a few crucial parts: we need to include separate DP parties to provide the input that must be adaptively secure and limit as much as possible their computational work, and in our protocol the parties jointly generate noise to satisfy a differential privacy guarantee. Beyond these modifications, the bulk of our contribution is a thorough proof of security of the protocol in the UC model (to make the proof go through, we needed to add re-encryption during the re-randomization phase), a specific application for the general theoretical protocol (making measurements in privacy-preserving systems like Tor), and a functional implementation of the protocol alongside empirical data measuring computation and communication costs gathered through experiments.

Several protocols to securely compute set-union cardinality (or the related problem of set-intersection cardinality) have been proposed [12, 18, 32, 48]. However, none of these provides all of the security properties that we desire for distributed measurement in a highly-adversarial setting: malicious security against a dishonest majority, forward privacy, and differentially-private output. Similar

protocols have been designed to securely compute set union [22, 28], but these protocols output every member of the union and not just its cardinality. General secure multiparty computation (MPC) protocols can realize any functionality including set-union cardinality, even in the UC model [1, 7], and recent advances in efficient MPC have been made in the multi-party, dishonest-majority setting that we require [10, 33, 34]. However, such protocols make use of a relatively expensive "offline" phase, while we intend to allow for measurements that are run on a continuous basis. We also consider a major advantage of PSC to be that it is conceptually straightforward and relies on a few well-understood tools.

Shuffles. The notion of a mixnet, which anonymizes a set of messages, was introduced by Chaum [8], and there are many schemes for proofs of shuffles [2, 13, 23, 27, 40], which have been developed as a fundamental primitive in privacy-preserving protocols. The first shuffles to be proved secure in the UC model are given by Wikström [51] and are designed to construct UC-secure mixnets. Mixnets and shuffling schemes have found a variety of applications, particularly anonymous communication [8, 11, 39, 49] and electronic voting [4, 29, 40].

8 CONCLUSION

Administrators of distributed systems have a critical need to understand how their systems are being used. Often, instrumenting the distributed system with measurement functionality and aggregating the result is a straightforward process. However, for systems in which privacy is paramount, maintaining and aggregating statistics is a far more complex task. Incorrect solutions can pose significant risk to the system's users.

This paper presents a protocol for private set-union cardinality (private set-union cardinality)—a zero-knowledge protocol for counting the unique number of items known across the system's nodes that ensures that no party can learn anything other than the cardinality. Our PSC protocol achieves private set-union cardinality using a combination of verifiable shuffles and differential privacy techniques. We formally prove the security of PSC using ideal functionalities under the UC framework.

We show that PSC can be practicality realized. Our implementation of PSC, which we release as free open-source software, demonstrates the scalability of our techniques. Running our protocol on 200,000 counters across 20 DPs with 5 CPs consumes only approximately 60 MB at each DP and approximately 2 GB traffic at each PSC-dedicated CP, making it practical for real-world deployments.

Importantly, PSC is a standalone protocol that is designed to be run alongside any distributed system in which there is a need to perform a private tabulation. As ongoing work, we are working towards deploying PSC at Tor's ingress points to discern the number of unique clients that are accessing the anonymity network. We envision that PSC is useful far beyond our own efforts, and could be applied to count unique visitors across websites, determine the number of users on a filesharing network (e.g., on a distributed hash table), quantify the number of duplicate voter registration records, among other uses.

Our implementation of PSC is available for download at http://safecounting.com/.

ACKNOWLEDGMENTS

We thank Adam O'Neill, Douglas Wikström, and the anonymous reviewers for helpful suggestions. This paper is partially funded from National Science Foundation (NSF) grants CNS-1527401 and CNS-1149832, the Defense Advanced Research Project Agency (DARPA), and the Department of Homeland Security (DHS) under agreement number FTCY1500057. The findings and opinions expressed in this paper are those of the authors and do not necessarily reflect the views of NSF, DARPA, or DHS.

REFERENCES

[1] G. Asharov and Y. Lindell. 2011. A Full Proof of the BGW Protocol for Perfectly-Secure Multiparty Computation. Cryptology ePrint Archive, Report 2011/136. (2011). http://eprint.iacr.org/2011/136.
[2] S. Bayer and J. Groth. 2012. Efficient Zero-knowledge Argument for Correctness of a Shuffle. In *Theory and Applications of Cryptographic Techniques (EUROCRYPT '12)*.
[3] A. Blum, K. Ligett, and A. Roth. 2008. A Learning Theory Approach to Non-interactive Database Privacy. In *Symposium on Theory of Computing (STOC '08)*.
[4] D. Boneh and P. Golle. 2002. Almost Entirely Correct Mixing With Application to Voting. In *Computer and Communications Security (CCS 2002)*. 68–77.
[5] F. Brandt. 2005. Efficient cryptographic protocol design based on distributed El Gamal encryption. In *International Conference on Information Security and Cryptology*. Springer, 32–47.
[6] R. Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Foundations of Computer Science (FOCS 2001)*.
[7] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. 2002. Universally Composable Two-Party and Multi-Party Secure Computation. In *Symposium on Theory of Computing (STOC)*.
[8] D. Chaum. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM* 24, 2 (1981), 84–90.
[9] D. Chaum and T. P. Pedersen. 1993. Wallet Databases with Observers. In *Advances in Cryptology (CRYPTO '92)*.
[10] I. Damgard, V. Pastro, N.P. Smart, and S. Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology (CRYPTO 2012)*.
[11] G. Danezis, R. Dingledine, and N. Mathewson. 2003. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Symposium on Security and Privacy*. 2–15.
[12] E. De Cristofaro, P. Gasti, and G. Tsudik. 2012. Fast and Private Computation of Cardinality of Set Intersection and Union. In *Cryptology and Network Security (CANS 2012)*.
[13] Y. Desmedt and K. Kurosawa. 2000. How to break a practical MIX and design a new one. In *Advances in Cryptology EUROCRYPT 2000*. Springer, 557–572.
[14] R. Dingledine, N. Mathewson, and P. Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium (USENIX)*.
[15] C. Dwork. 2006. Differential Privacy. *Automata, Languages and Programming* (2006), 1–12.
[16] C. Dwork, F. McSherry, K. Nissim, and A. Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography Conference (TCC)*.
[17] C. Dwork, A. Roth, and others. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014), 211–407.
[18] R. Egert, M. Fischlin, D. Gens, S. Jacob, M. Senker, and J. Tillmanns. 2012. Privately Computing Set-Union and Set-Intersection Cardinality via Bloom Filters. In *Information Security and Privacy (ACISP 2015)*.
[19] T. Elahi, G. Danezis, and I. Goldberg. 2014. PrivEx: Private Collection of Traffic Statistics for Anonymous Communication Networks. In *ACM Conference on Computer and Communications Security (CCS)*.
[20] T. ElGamal. 1985. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE transactions on Information Theory* 31, 4 (1985), 469–472.
[21] A. Fiat and A. Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology (CRYPTO '86)*.
[22] M. J Freedman, K. Nissim, and B. Pinkas. 2004. Efficient Private Matching and Set Intersection. In *Advances in Cryptology (Eurocrypt)*.
[23] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, and K. Sako. 2003. An Implementation of a Universally Verifiable Electronic Voting Scheme Based on Shuffling. In *Financial Cryptography (FC'02)*.
[24] J. A. Garay, P. MacKenzie, and K. Yang. Strengthening Zero-knowledge Protocols Using Signatures. In *Theory and Applications of Cryptographic Techniques (EUROCRYPT '03)*.

[25] O. Goldreich. 2001. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press.
[26] S. Goldwasser and Y. Lindell. 2005. Secure Multi-Party Computation without Agreement. *Journal of Cryptology* 18, 3 (2005), 247–287.
[27] J. Groth. 2003. A Verifiable Secret Shuffle of Homomorphic Encryptions. In *Theory and Practice in Public Key Cryptography (PKC '03).*
[28] C. Hazay and K. Nissim. 2012. Efficient Set Operations in the Presence of Malicious Adversaries. *Journal of Cryptology* 25, 3 (July 2012), 383–433.
[29] M. Jakobsson, A. Juels, and R. L. Rivest. 2002. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX Security Symposium.*
[30] R. Jansen and A. Johnson. 2016. Safely Measuring Tor. In *ACM Conference on Computer and Communications Security (CCS).*
[31] S. P. Kasiviswanathan and A. Smith. 2014. On the 'Semantics' of Differential Privacy: A Bayesian Formulation. *Journal of Privacy and Confidentiality* 6, 1 (2014).
[32] L. Kissner and D. Song. 2005. Privacy-Preserving Set Operations. In *Proceedings on Advances in Cryptology (CRYPTO 2005).*
[33] E. Larraia, E. Orsini, and N.P. Smart. 2014. Dishonest Majority Multi-Party Computation for Binary Circuits. In *Advances in Cryptology (CRYPTO 2014).*
[34] Y. Lindell, B. Pinkas, N.P. Smart, and A. Yanai. 2015. Efficient Constant Round Multi-party Computation Combining BMR and SPDZ. In *Advances in Cryptology (CRYPTO 2015).*
[35] K. Loesing, S. J Murdoch, and R. Dingledine. 2010. A Case Study on Measuring Statistical Data in the Tor Anonymity Network. In *Financial Cryptography and Data Security (FC).*
[36] A. Mani and M. Sherr. 2017. Historε: Differentially Private and Robust Statistics Collection for Tor. In *Network and Distributed System Security Symposium (NDSS).*
[37] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker. 2008. Shining Light in Dark Places: Understanding the Tor Network. In *Privacy Enhancing Technologies Symposium (PETS).*
[38] F. McSherry and K. Talwar. 2007. Mechanism design via differential privacy. In *Foundations of Computer Science (FOCS'07).*
[39] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. 2003. Mixmaster Protocol — Version 2. IETF Internet Draft. (July 2003).
[40] C. Andrew Neff. 2001. A Verifiable Secret Shuffle and its Application to E-voting. In *ACM Conference on Computer and Communications Security (CCS).*
[41] L. Nguyen, R. Safavi-Naini, and K. Kurosawa. Verifiable Shuffles: A Formal Model and a Paillier-based Efficient Construction with Provable Security. In *Applied Cryptography and Network Security (ACNS).* Springer, 61–75.
[42] K. Nissim, S. Raskhodnikova, and A. Smith. 2007. Smooth Sensitivity and Sampling in Private Data Analysis. In *Symposium on Theory of Computing (STOC '07).*
[43] NIST 1999. Recommended Elliptic Curves for Federal Government Use. (1999). Available at http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf.
[44] C. P. Schnorr. 1991. Efficient Signature Generation by Smart Cards. *Journal of Cryptology* 4, 3 (1991).
[45] V. Shoup. 2004. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology EPrint Archive* 2004 (2004), 332.
[46] C. Soghoian. 2011. Enforced Community Standards For Research on Users of the Tor Anonymity Network. In *Workshop on Ethics in Computer Security Research (WECSR).*
[47] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. 2001. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM).*
[48] J. Vaidya and C. Clifton. 2005. Secure Set Intersection Cardinality with Application to Association Rule Mining. *J. Comput. Secur.* 13, 4 (2005).
[49] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. 2015. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Operating Systems Principles.* ACM, 137–152.
[50] K. Varda. 2008. Protocol buffers: Google's data interchange format. *Google Open Source Blog, Available at least as early as Jul* (2008).
[51] D. Wikström. 2005. A Sender Verifiable Mix-Net and a New Proof of a Shuffle. Cryptology ePrint Archive, Report 2005/137. (2005). http://eprint.iacr.org/2005/137.

A IDEAL FUNCTIONALITIES

For all ideal functionalities we assume the group G and generator g used throughout the protocol are publicly known. The Zero-Knowledge proofs compiled from Σ-protocols are converted to UC-secure ideal functionalities in the CRS model as described by Hazay and Nissim[28].

A.1 ZKP of Re-encryption Re-randomization

Suppose we have a ciphertext pair (A, B) and we wish to present (α, β) as a re-encryption re-randomization of it – that is, $\alpha = (Ag^s)^q$, $\beta = (By^s)^q$ for some random shift value s and re-randomization value q. Then $A, B, \alpha, \beta, g, y$ are known to both the prover and the verifier. We describe the proof:

(1) The Prover P selects t_1, t_2 at random and sends $T_1 = A^{t_1}g^{t_2}$, $T_2 = B^{t_1}y^{t_2}$
(2) The Verifier V sends a random challenge c to P.
(3) The Prover sends $r_1 = qc + t_1$, $r_2 = sqc + t_2$
(4) The Verifier accepts the proof iff $A^{r_1}g^{r_2} = \alpha^c T_1$ and $B^{r_1}y^{r_2} = \beta^c T_2$

We claim the above proof is an HVZK proof that proves knowledge of q, s such that the equations above for α, β hold.

Proof:

(1) **Completeness.** P generates t, s, q and learns c so it can generate r_1,r_2. We verify the two equations:

$$A^{r_1}g^{r_2} = A^{qc+t_1}g^{r_2} = A^{qc+t_1}g^{sqc+t_2} = A^{qc}g^{sqc}A^{t_1}g^{t_2} = \alpha^c T_1$$
$$B^{r_1}y^{r_2} = B^{qc+t_1}y^{r_2} = B^{qc+t_1}y^{sqc+t_2} = B^{qc}y^{sqc}B^{t_1}y^{t_2} = \beta^c T_2$$

(2) **Special Soundness.** Suppose the prover provides two proofs with the same commitment values t_1, t_2, with challenges c_1 and c_2. Then we have:

$$r_1 = qc_1 + t_1 \qquad r_1' = qc_2 + t_1$$
$$r_2 = sqc_1 + t_2 \qquad r_2' = sqc_2 + t_2$$

Then it is easy to see that $q = \frac{r_1 - r_1'}{c_1 - c_2}$ and then $s = \frac{r_2 - r_2'}{q(c_1 - c_2)}$ so that special soundness is satisfied.

(3) **Honest Verifier Zero Knowledge.** We define a simulator as follows: The simulator runs the prover as normal until it receives c, then rewinds V, selects random r_1, r_2 and sets

$$T_1 = \frac{A^{r_1}g^{r_2}}{\alpha^c} \qquad T_2 = \frac{B^{r_1}y^{r_2}}{\beta^c}$$

Since V is an honest verifier, it, given the same randomness, provides the same challenge c and the equations required for V to verify both hold and the simulation is successful.

$\mathcal{F}_{ZKP-RR}$

(1) Upon receiving $(\text{PROVE}, (\vec{A}, \vec{B}), (\vec{\alpha}, \vec{\beta}), \vec{q}, \vec{s}, P_i, P_j))$ from P_i for $(\vec{A}, \vec{B})$ and $(\vec{\alpha}, \vec{\beta})$ vectors of ciphertexts, $\vec{s}, \vec{q}$ vectors in $\mathbb{Z}_p$, set R to 1 if for every i,

$$\alpha_i = (A_i g^{s_i})^{q_i} \qquad \beta_i = (B_i Y^{s_i})^{q_i}$$

and set $R = 0$ otherwise.
(2) Send $(\text{RESULT}, (\vec{A}, \vec{B}), (\vec{\alpha}, \vec{\beta}), P_i, P_j)$ to S, P_j.

Figure 7: $\mathcal{F}_{ZKP-RR}$, the ideal functionality for a re-encryption re-randomization ZKP

A.2 ZKP of a Re-encryption shuffle

This is the idealization of our re-encryption shuffle ZKP. This functionality has been realized in the UC model by Wikström [51], specifically in Appendix F. We allow the functionality to take the permutation explicitly as an input for convenience.

$\mathcal{F}_{ZKP-S}$

(1) Upon receiving $(PROVE, (\vec{A}, \vec{B}), (\vec{\alpha}, \vec{\beta}), \vec{r}, \pi, y, P_i, P_j)$ from P_i for ciphertext vectors $(\vec{A}, \vec{B}), (\vec{\alpha}, \vec{\beta})$, vector of group elements $\vec{r}$, public key y and π a permutation of $\{1, ..., n\}$.
(2) Set R to 1 if, for each i, $(\alpha_{\pi(i)}, \beta_{\pi(i)})$ is a re-encryption of (A_i, B_i) under public key y using random element r_i. Set R to 0 otherwise.
(3) Send $(RESULT, (\vec{A}, \vec{B}), (\vec{\alpha}, \vec{\beta}), y, P_i, P_j, R)$ to S, P_j.

Figure 8: $\mathcal{F}_{ZKP-S}$, the ideal functionality for a re-encryption shuffle ZKP

A.3 ZKP of Knowledge of Discrete Log

The prover wants to prove knowledge of x such that $y = g^x$ so we use the Σ−protocol described by Schnorr [44]. We describe the protocol and since it is standard we omit the proof:

(1) The Prover P selects t at random in G and sends $T = g^t$
(2) The Verifier selects a random challenge c to P.
(3) The Prover sends $s = t + cx$
(4) The Verifier accepts the proof if and only if $g^s = Ty^c$

$\mathcal{F}_{ZKP-DL}$

(1) Upon receiving (PROVE, x, y, P_i, P_j) for $x \in \mathbb{Z}_p$, $y \in G$, set R to 1 if $y = g^x$, 0 otherwise.
(2) Send (RESULT, y, P_i, P_j, R) to S, P_j.

Figure 9: $\mathcal{F}_{ZKP-DL}$, the ideal functionality for a zero knowledge proof of knowledge of a discrete log

A.4 ZKP of Knowledge of Equality of Discrete Logs

Suppose we have a ciphertext pair, (A, B). We want to present (A', B') and a proof that there exists some r such that $A' = A^r$, $B' = B^r$, or that $\log_A A' = \log_B B' = r$ where r is known to the prover. A Σ-protocol for this proof is described by Chaum and Pedersen [9]. We describe the protocol and since it is standard we omit the proof:

(1) The Prover P selects t at random and sends $T_1 = A^t, T_2 = B^t$ to V.
(2) The Verifier sends a random challenge c to P.
(3) The Prover sends $s = rc + t$ to V
(4) The Verifier accepts the proof iff $A^s = A'^c T_1$ AND $B^s = B'^c T_2$

$\mathcal{F}_{ZKP-DLE}$

(1) Upon receiving (PROVE, $(\vec{A}, \vec{B}), (\vec{\alpha}, \vec{\beta}), x, P_i, P_j$) from P_i, for $(\vec{\alpha}, \vec{\beta})$ a vector of ciphertext pairs and $x \in \mathbb{Z}_p$, set R to 1 if for every i, $\log_{\beta_i} \alpha_i = \log_{B_i} A_i = x$ and set $R = 0$ otherwise.
(2) Send (RESULT, $(\vec{A}, \vec{B}), (\vec{\alpha}, \vec{\beta}), P_i, P_j, R$) to S, P_j.

Figure 10: $\mathcal{F}_{ZKP-DLE}$, the ideal functionality for a zero knowledge proof of knowledge of the equality of two discrete logs

A.5 Ideal Functionality for Authenticated Broadcast

We define an ideal functionality for authenticated broadcast with abort and no fairness. We use a minor modification of the protocol defined in [26].

Broadcast with Abort

- P_i has input x and sends x to all parties.
- Upon receiving a value x^j from P_i, party P_j sends the value x^j to all other parties.
- Party P_j waits to receive a message from each party, and checks to see if these match the message sent by P_i. If so, output this message. If not, output ABORT.

This is identical to the UC secure protocol which realizes authenticated broadcast in [26] except that instead of doing nothing upon receiving mismatched messages, the parties output ABORT. The proof that the protocol above UC-realizes the ideal functionality below is almost identical to the proof given of broadcast with abort and no fairness, where the simulator instead instructs honest parties who receive mismatched values to abort through the ideal functionality instead of giving them no output. Honest parties in our protocol who receive ABORT from $\mathcal{F}_{BC}$ output ABORT and terminate.

$\mathcal{F}_{BC}$

(1) Upon receiving a message (m) from some party P, send (m,P) to S.
(2) Upon receiving (DELIVER, [IDs]) from S, send (m, P) to the parties indicated by [IDs]
(3) If any party is corrupted, upon receiving (ABORT, [IDs]) from S, send (ABORT) to the parties indicated by [IDs]

Figure 11: $\mathcal{F}_{BC}$, the ideal functionality for authenticated broadcast with abort

A.6 Secure Communication

This ideal functionality directly defines our communication model between DPs and CPs, and we take it directly from Canetti[7].

$\mathcal{F}_{SC}$

(1) Upon receiving input (m, S, R) for some message m and two parties S, R from S, send $(\text{SENT}, |m|, \text{S}, \text{R})$ to S.
(2) When S instructs, send (m, S) to R.

Figure 12: $\mathcal{F}_{SC}$, the ideal functionality for secure point-to-point communication

A.7 PSC

We repeat the definition of the the ideal functionality for PSC from Figure 2, parameterized by b, n bits of data and noise each

$\mathcal{F}_{PSC}$

(1) Wait until $(\text{CORRUPT-CP}, C)$ is received from A. For each $i \in C$, add CP_i to the list of corrupted parties.
(2) Upon receiving $(\text{CORRUPT-DP}, i)$ from A, add DP_i to the list of corrupted parties. Note that past inputs to DP_i are not revealed to A upon corruption.
(3) Set $v^i \leftarrow 0^b$ for each DP_i. Upon receiving $(\text{INCREMENT}, k)$ from DP_i, set $v^i_k \leftarrow 1$.
(4) Upon receiving GO from DP_i, ignore further messages from DP_i, and send (GO, DP_i) to A.
(5) Upon receiving GO from CP_i, ignore further messages from CP_i, and send (GO, CP_i) to A.
(6) Set $v^A \leftarrow 0^b$. Upon receiving $(\text{INCREMENT}, k)$ from A, if any CP or DP is corrupted, set $v^A_k \leftarrow 1$.
(7) Sample the noise value as $N \sim \text{Bin}(n, 1/2)$, where $\text{Bin}(n, p)$ is the binomial distribution with n trials and success probability p.
(8) Let $w_i = \max\left(v^1_i, \ldots, v^d_i, v^A_i\right)$, $1 \le i \le b$. Compute output $z \leftarrow \sum_{i=1}^{b} w_i + N$. Once GO has been received from each CP, output z to CP_m.
(9) If at any point $\mathcal{F}_{PSC}$ receives (ABORT, IDS) from A, send ABORT to all parties indicated by IDS and do not send additional outputs besides other ABORT commands as directed by A.

Figure 13: $\mathcal{F}_{PSC}$, the Private Set-Union Cardinality ideal functionality

B PROOFS

B.1 Preliminaries

We deal extensively with vectors of ciphertexts and plaintexts and write them $\vec{v}$. We write an individual component k of list $\vec{v}$ as v_k.

We assume all parties agree on a session ID which is a prefix of all messages in the protocol (or else the parties ignore them), and that they agree on the publicly known group and protocol parameters g, G, b, n.

We assume perfectly secure point to point channels described explicitly in Appendix A.6 that leak only the length of each message between each individual parties, though the only point to point communication that occurs in the protocol is directly from DPs to the CPs to submit initial and final shares, which in the implementation is protected through a symmetric encryption scheme that we choose not to model explicitly. We assume these channels are point-to-point secure and implicit in the communication model, realized through $\mathcal{F}_{SC}$ and do not mention the functionality again, including it only for completeness.

While the CPs execute a single of sequence of computations with no delay during the run of the protocol, the DPs keep blind counters and collect data for some significant period of time, and may, during that time be compromised by some attacker so while we consider static corruptions to be reasonable to model the behavior of the CPs, it is an important security property of our protocol that if a DP is compromised while it is collecting data, the protocol remains correct and the privacy of the inputs up to that point remains preserved.

In order to do this we work in the erasure model and allow for the DPs to securely erase records of their behavior.

So our corruption model is static corruptions with respect to the CPs, and adaptive with respect to the DPs.

B.2 Proving the simulator correct

We define the simulator S defined in Figure 14. Fix a probabilistic polynomial time environment Z and assume a dummy adversary A. S runs a copy of A and simulates the other parties for A's benefit, forwarding messages from Z to its simulated A and back. We define CP_h to be the last honest CP. We note that S has access to the randomness used for every ciphertext of the CPs as it is provided to an appropriate ZKP functionality, and that S knows the private keys for each CP (it generates the keys for the honest ones, and the corrupt ones send the private keys to $\mathcal{F}_{DL}$ to prove they are not related to previous keys).

In the following when we say CP_h 'decrypts' a vector of plaintexts $\vec{P}$ we mean that CP_h takes the first component in each vector presented to it, α_i and broadcasts a vector of the pairs:

$$(\alpha_i, \alpha_i^{x_{h+1}+\ldots+x_m} g^{p_i})$$

or if $h = m$, a vector of (α_i, g^{p_i}) and proves this operation correct by instructing $\mathcal{F}_{DL}$ to tell every other CP that the decryption was done correctly.

We need to show that Z cannot with non negligible probability distinguish between interacting with A in the ($\mathcal{F}_{ZKP-RR}$, $\mathcal{F}_{ZKP-DL}$, $\mathcal{F}_{ZKP-DLE}$, $\mathcal{F}_{BC}$, $\mathcal{F}_{ZKP-S}$)-hybrid execution and interacting with S in an ideal execution where S has access to only $\mathcal{F}_{PSC}$.

We define the following sequence of (sequences of) hybrid executions which each run Z and output the output of Z. We assume a dummy adversary A and prove there is no PPT algorithm Z which

Simulation of Aborts. Simulation of aborts: if at any time, an honest CP aborts, the simulator must send (ABORT, ID) to $\mathcal{F}_{PSC}$ where ID is the identity of the aborting party.

Extraction. Set CP_h as the last honest CP. S instructs every ZKP functionality to respond to any proof attempt by CP_h with 1, indicating the proof was correct. S runs simulated honest DPs honestly, and since they are not connected to the environment Z they receive no INCREMENT instructions and so always input 0. When S receives (GO, DP_i) for some honest DP_i, DP_i proceeds as if it had received GO from Z. CP_h submits an encryption of 0 during data submission. If any party is corrupted S submits data to $\mathcal{F}_{PSC}$ as follows: After the CPs have aggregated their submissions into the vector $\vec{c}$, S recovers the plaintexts using the randomness submitted to $\mathcal{F}_{DL}$ and combines it with the values sent to CP_h by the DPs into a vector $\vec{v}$ sends (INCREMENT,k) to $\mathcal{F}_{PSC}$ for each nonzero bin v_k, then submits GO on behalf of each honest DP.

Computations. Now S waits for $\mathcal{F}_{PSC}$ to output its result r and recovers it, and delays any messages from $\mathcal{F}_{PSC}$ to other parties besides aborts. For the remainder, the simulator deviates from running the honest participants and ideal functionalities honestly in the following ways:

(1) During noise generation, instead of CP_h presenting a correct re-encryption permutation of the two values, present two random encryptions of 2
(2) During re-encryption permutation, instead of CP_h presenting a correct re-encryption permutation of the vector it was given, present in each bin k an encryption of k
(3) During re-encryption re-randomization, instead of CP_h presenting a correct re-encryption re-randomization of the vector it was given, present in each bin k an encryption of k
(4) During decryption, generate a fake vector $\vec{f}$ of exponential ElGamal plaintexts with r uniformly random nonzero elements, then $(b+n)-r$ zeros. Generate a uniformly random permutation π and set $\vec{f} = \pi(\vec{f})$. Then if $h = m$, present $\vec{f}$ on behalf of CP_h. Otherwise, for each bin k, write the first component of bin k of the vector presented to CP_h as α_k, write x_i as the private key of CP_i sent to $\mathcal{F}_{DL}$ during key generation and present the following vector of ciphertexts: for each component k, present
$$(\alpha_k, \alpha_k^{x_{h+1}+\ldots+x_m} f_k)$$
on behalf of CP_h.

Figure 14: The Ideal Model Simulator S

can distinguish between each adjacent pair of hybrids with non-negligible probability.

B.2.1 Aborts. If any honest SK aborts at any phase in the protocol, the future phases do not take place and the view of Z terminates in the ideal execution, ($\mathcal{F}_{ZKP-RR}$, $\mathcal{F}_{ZKP-DL}$, $\mathcal{F}_{ZKP-DLE}$, $\mathcal{F}_{BC}$, $\mathcal{F}_{ZKP-S}$)-hybrid execution, and every intermediate hybrid machine: in the hybrid execution, the honest CP terminates, and in the ideal execution S observes this behavior and sends an appropriate ABORT message to $\mathcal{F}_{PSC}$, terminating the protocol. Then in the following discussion, we consider only executions where Z does not behave in a manner which causes an abort, and note that any execution where Z does is a prefix of the one we consider below where the protocol completes; then by showing two protocol-completing hybrids are indistinguishable, we can infer two equal-length prefixes of them are indistinguishable as well.

We define a sequence of hybrids. First let D_0 be the ($\mathcal{F}_{ZKP-RR}$, $\mathcal{F}_{ZKP-DL}$, $\mathcal{F}_{ZKP-DLE}$, $\mathcal{F}_{BC}$, $\mathcal{F}_{ZKP-S}$)-hybrid execution.

B.2.2 Data bins. We define the following hybrids D_i for $1 \leq i \leq b$ where the increment values given to honest DPs are inserted directly by CP_h. More specifically, D_i is defined exactly as the ($\mathcal{F}_{ZKP-RR}$, $\mathcal{F}_{ZKP-DL}$, $\mathcal{F}_{ZKP-DLE}$, $\mathcal{F}_{BC}$, $\mathcal{F}_{ZKP-S}$)-hybrid execution D_0 except as follows:

D_i

(1) For any honest DP do not increment any bin $k \leq i$ even if instructed to.
(2) Initialize a vector of length k of all zeros, $\vec{h}$ and if an honest CP receives INCREMENT for bin k, increment h_k for that bin by a random value.
(3) For every $k \leq i$, add values submitted by each DP to CP_h in bin k to h_k and submit h_k on behalf of CP_h.

Figure 15: D_i, hybrids with simulated DPs

We claim that the hybrids $D_0, D_1, \ldots, D_b$ are identically distributed.

LEMMA B.1. *D_i and D_{i+1} are identically distributed.*

The difference between these two hybrids is that the incrementing of bin $i+1$ is done by honest DPs in D_i and sent to CP_h to forward during data submission and is done by an honest CP_h directly in D_{i+1}, but communication between these two parties is perfectly secure so that if only honest DPs which are never corrupted increment bin $i+1$, this is simply a view change.

If no honest DP is instructed to increment bin $i+1$, D_i and D_{i+1} are defined identically.

So consider the case that some honest DP_j has been corrupted after it has incremented bin $i+1$.

We can assume without loss of generality Z knows the shares in bin $i+1$ from DP_j for every CP besides CP_h's first share from DP_j. Call this share s_h and suppose again without loss of generality that Z has corrupted DP_j before it submits its second shares so that Z may select them instead of after where it may simply observe them.

While Z only ever observes the encrypted submission of CP_h, we prove something stronger: that the plaintext p of the value submitted by CP_h in both hybrids is identically distributed.

We show directly that every value in $\mathbb{Z}_p$ is equally likely to be submitted by CP_h in both hybrids. Consider D_i and fix some $k \in \mathbb{Z}_q$. CP_h submits in this hybrid $p = s_h + s'_h$ where s_h is its original blind share, and s'_h is the final one selected by Z. s_h is generated by an honest DP, sent securely to CP_h. In this case by assumption the honest DP has incremented its blind counter T^j_{i+1} and securely erased everything but this so T^j_{i+1} is random and independent from each blind share, including s_h. Therefore s'_h is selected by Z with no knowledge about s_h: Z knows the sum of s_h and original value of T^j_{i+1}, but has no information about either individual quantity. So the likelihood CP_h submits k is the probability that $s_h = k - s'_h$ with s_h random, which is $\frac{1}{|\mathbb{Z}_q|}$.

Now consider D_{i+1}. Fix s_h, let Z select any s'_h. The hybrid selects a value v completely independent of Z and CP_h submits $s_h + v + s'_h$ so the likelihood CP_h submits k is the probability that $v = k - s_h - s'_h$ which is $\frac{1}{|\mathbb{Z}_q|}$.

Then we have the sequence of hybrids: $D_0, D_1, ..., D_b$ are identically distributed.

B.2.3 Data Bins 2. We define a series of hybrids D^2_i where for bins $1 \le i \le b$, CP_h submits an encryption of 0. More specifically, D^2_i is defined exactly as D_b except as follows:

D^2_i

(1) During data submission, for bins $k \le i$, CP_h submits an encryption of 0 instead of the correct value.
(2) During decryption for CP_h, instead of decrypting bins $k \le i$ correctly, decrypt p_k, defined as follows: extract the submissions from every CP and sum them, along with h_k, then multiply the sum by the re-randomization factors $q_1 ... q_m$ provided by the CPs to $\mathcal{F}_{ZKP-RR}$.

Figure 16: D^2_i, hybrids with simulated data submission

Set $D^2_0 = D_b$. We wish to prove a sequence of hybrid executions, $D^2_0, D^2_1, ..., D^2_b$ are indistinguishable from each other.

We define the following modified *IND-CPA* game and follow the method described by Shoup [45].

Game 1

$x_1 \xleftarrow{R} \mathbb{Z}_q$, $y_1 \leftarrow g^{x_1}$, $r \xleftarrow{R} \mathbb{Z}_q$
$y_2 \leftarrow A(y_1)$, $y \leftarrow y_2 y_1$
$b \xleftarrow{R} \{0, 1\}$
$m_0, m_1 \leftarrow A(y)$, $\alpha \leftarrow g^r$, $\beta \leftarrow y^r g^{m_b}$
$\hat{b} \leftarrow A(y, \alpha, \beta)$
Event S_1 is the event that $b = \hat{b}$.

Figure 17: Game 1, a modification of the IND-CPA game for ElGamal

Lemma B.2. *For any PPT adversary A, $|P(S_1) - 1/2| < \epsilon$ for ϵ negligible.*

Since this game allows the adversary to control a portion of the public key, it is not exactly the standard *IND-CPA* game for ElGamal so we provide a direct proof.

Distinguisher 1

$\delta_1(\eta, \nu, \rho)$
$y_1 \leftarrow \eta$
$y_2 \leftarrow A(y_1)$
$b \xleftarrow{R} \{0, 1\}$
$m_0, m_1 \leftarrow A(y)$, $\alpha \leftarrow \nu$, $\beta \leftarrow \rho g^{m_b}$
$\hat{b} \leftarrow A(y, \alpha, \beta)$
Output $b == \hat{b}$

Figure 18: The distinguisher for Game 1

Now we note if (η, ν, ρ) is a DDH tuple, then α, β is an encryption of m_b for some random b just as in Game 1, whereas if (η, ν, ρ) is not a DDH tuple, β is uniformly random and independent from b since ρ is, which means any algorithm playing this game has zero advantage. So for uniformly random independent z_1, z_2, z_3:

$$|P(\delta(g^{z_1}, g^{z_2}, g^{z_1 z_2}) = 1) - P(\delta(g^{z_1}, g^{z_2}, g^{z_3}) = 1)| < \epsilon$$

where ϵ is negligible for any algorithm A by the DDH assumption. And in particular for $A = \delta$ this gives $|P(S_1) - 1/2|$ is negligible.

Lemma B.3. *If there exists some Z that has non negligible advantage distinguishing between D^2_i and D^2_{i+1}, there exists an adversary for Game 1 which has non-negligible advantage.*

We play Game 1. Our distinguisher accepts y_1 from Game 1 and is expected to produce y_2. It begins the protocol, broadcasting y_1 as the public key of CP_h, and lets

$$y_2 = \prod_{k \neq h} g^{x_k}$$

so that y in Game 1 is the joint shared public key for the protocol execution.

Then give $m_0 \leftarrow 0, m_1 \leftarrow p_{i+1}$ as defined above to the challenger in Game 1 and submit (α, β) as the submission for CP_h for bin $i + 1$.

Then we respond to the challenger in Game 1 with the response of Z in the protocol execution. We note that if $b = 0$, Z is interacting with D^2_i whereas if $b = 1$, Z is interacting with D^2_{i+1}. So Z's advantage in distinguishing the two hybrids is equal to our advantage in Game 1, which must be negligible.

The decrypted value in both hybrids is p_k or the same encryption of it.

So we have a sequence of hybrids $D^2_0, ...D^2_b$ which are adjacent-wise indistinguishable.

B.2.4 Noise Bins. We define a hybrid for each $1 \le i \le n$ where the noise ciphertexts are replaced by dummy ciphertexts. More specifically, N_i is defined exactly as D^2_b except as follows:

N_i

(1) For $k \leq i$, during noise generation for bin k, CP_h presents two random encryptions of 2 instead of an encryption of 0 and 1.
(2) During decryption, for each bin k generate a random permutation π_h^k and calculate the plaintext p_k for noise bit k using permutation π_h^k and re-randomization factors $q_1 ... q_m$, then decrypt p_k on behalf of CP_h

Figure 19: N_i the hybrids where we replace noise calculations

Lemma B.4. *If there exists some Z that has non negligible advantage distinguishing between N_i and N_{i+1}, there exists an adversary for Game 1 which has non-negligible advantage.*

We play Game 1 and consider a hybrid which only changes the first bin, N'_{i+1}:

As before, set y_1 as the public key of CP_h, and give the challenger y_2 so that the joint public key in the execution is y in Game 1. Then set $m_0 = 2$, and set m_1 as the plaintext in the first ciphertext given to CP_h for noise generation bit $i+1$. Then replace the first ciphertext with α, β from the challenger in Game 1. Output the output of Z.

We note that if $b = 1$, then this is identically N_i. If $b = 0$,then this is N'_i. So that if Z can distinguish between N_i, N'_i with non-negligible probability, we have constructed an adversary for Game 1 which succeeds with non-negligible probability which contradicts lemma B.2.

Then consider the transition from N'_i to N_{i+1}. We play the identical game but replace the second bin as well and the argument is identical, so that we have a sequence: N_i, N'_i, N_{i+1} of hybrids which are adjacentwise indistinguishable for every i, so we have a sequence of hybrids: $D_b^2 = N_0, N'_0, N_1, N'_1, ..., N_n$

B.2.5 Noise Bins 2. We define the following hybrid which independently generates the noise bits according to a fixed distribution independent of Z during decryption. More specifically, N_n^2 is defined exactly as N_n except as follows:

N_n^2

(1) During decryption, for each $k \leq n$, generate b_k uniformly in $\{0, 1\}$. Then on behalf of CP_h present an encryption of $b_k q_1 q_2 ... q_m$ to the remaining CPs where q_i is the re-randomization factor provided by CP_i.

Figure 20: The noise view-change hybrids

Lemma B.5. *N_n^2 and N_n are identically distributed.*

The determination of the plaintext in each execution is done by random selection independent of Z with the same distribution: In N_n we select, for each k, $\pi_h \xleftarrow{R} \{(1), (12)\}$ (where we define permutations in cyclic notation) and in N_n^2 we select $b_k \xleftarrow{R} \{0, 1\}$. Fix the parity of the number s of non-identity permutations selected by Z for a given noise bin k in a given execution. Then if $s = 0$, selecting $\pi_h = (1)$ is selecting $b_k = 0$, whereas selecting $\pi_h = (12)$ is selecting $b_k = 1$, and if $s = 1$ the choices are reversed. The distributions in each case are identical, which is the only difference between N_n and N_n^2.

B.2.6 Permutation. We define the following sequence of hybrids for $1 \leq i \leq b + n$ where we replace each permutation ciphertext with a dummy ciphertext. More specifically, P_i is defined exactly as N_n^2 except as follows:

P_i

(1) During re-encryption permutation for CP_h, generate a permutation π_h and for bins $k \leq i$, replace the ciphertext in bin k with a random encryption of k. For bins $k > i$ permute and re-encrypt honestly according to π_h.
(2) During decryption for CP_h, for bins $k \leq i$ calculate the final plaintexts as if π_h were applied to the ciphertexts by CP_h. Decrypt these values on behalf of CP_h.

Figure 21: P_i, hybrids where the permutations are simulated

Let $P_0 = N_n^2$.

Lemma B.6. *If there exists some Z that has non negligible advantage distinguishing between P_i and P_{i+1}, there exists an adversary for Game 1 which has non-negligible advantage.*

Fix an i and suppose there exists such a Z. We play Game 1.

Receive y_1 from the challenger in Game 1 and present it as the public key of CP_h. Give y_2 as the joint public key of the remaining CPs so that y in Game 1 is the joint public key for the protocol. Run the protocol as in N_n^2 until the re-encryption permutation for CP_h. Select a random permutation π_h and determine the vector v an honest CP_h would present as a re-encryption permutation. Set $m_0 = v_{i+1}$ where v_{i+1} is the plaintext in bin $i + 1$ of vector v. Set $m_1 = i + 1$. Generate the vector for CP_h to broadcast defined componentwise as follows: For components $k < i + 1$, present a random encryption of k. For component $i + 1$, present α, β from the challenger in Game 1. For $k > i + 1$, present an encryption of v_k as in N_n^2.

During decryption, calculate the plaintext that would have been generated using re-randomization factors $q_1, ..., q_m$ and the permutations and re-encryption factors provided by all other CPs and π_h. Present the encryption of these plaintexts for the remaining CPs (or the plaintexts, if $m = h$) and give the challenger in Game 1 the output of Z.

Now only the value in bin $i + 1$ changes, and that if b in Game 1 is 0, then α, β is the correct value for bin $i + 1$ for an honestly generated re-encryption permutation, so this is exactly the value in bin $i + 1$ for P_i. If b in Game 1 is 1, then it is an encryption of $i + 1$ which is exactly the value for bin $i + 1$ in P_{i+1}.

Then if Z can distinguish between P_i and P_{i+1} with non negligible advantage, then the scheme above is an adversary for Game 1 with non negligible advantage, which is a contradiction by Lemma B.2.

So we have the sequence of hybrids: $N_n^2 = P_0, P_1, ..., P_{b+n}$ which are adjacentwise indistinguishable.

B.2.7 Permutation 2. Now we define the hybrid P^2_{b+n} which generates a permutation at random independent from the view of Z to apply to the plaintexts. More specifically, P^2_{b+n} is defined exactly as P_{b+n} except as follows:

P^2_{b+n}

(1) During decryption, sort all of the plaintexts in the vector presented to CP_h, then shuffle them according to a random permutation π_h.
(2) For each bin, present an encryption of the shuffled plaintexts behalf of CP_h to the remaining CPs

Figure 22: P^2, permutation view-change hybrids

In P_{b+n} CP_h submits no information about any permutation to Z since all the values are dummy values. More explicitly, consider the composed permutation of the plaintexts in both hybrids.

In P_{b+n}, the permutation is some composition $\pi_{P1} = \pi_1 \circ \pi_h \circ \pi_2$ where without loss Z has selected π_1, π_2 (letting π_1 or π_2 be the identity if $h = 1$ or $h = m$), and π_h is random.

Whereas in P^2_{b+n}, the permutation is $\pi_{P2} = \pi_s \circ \pi$ where π_s is the permutation which sorts the plaintexts, then π is random.

Lemma B.7. *π_{P1} and π_{P2} are identically distributed.*

Each permutation π is selected in P^2_{b+n} with probability $\frac{1}{(b+n)!}$ and for each π there exists a unique π_h such that $\pi_{P1} = \pi_{P2}$:

Every permutation is invertible so we let

$$\pi_h = \pi_1^{-1} \circ \pi_s \circ \pi \circ \pi_2^{-1}$$

and note that the probability this π_h is selected in P_{b+n} is $\frac{1}{(b+n)!}$ which is the same as in P^2_{b+n}.

Corollary B.8. *P_{b+n} and P^2_{b+n} are identically distributed*

Since only these permutations are changed between P_{b+n}, P^2_{b+n} and the permutations are identically distributed, the hybrids themselves are identically distributed.

B.2.8 Re-randomization. We define the following sequence of hybrids for $1 \leq i \leq b + n$ which replace each ciphertext during re-randomization with a dummy. More specifically, R_i is defined exactly as P^2_{b+n} except as follows: We let $R_0 = P^2_{b+n}$.

Lemma B.9. *If there exists some Z that has non negligible advantage distinguishing between R_i and R_{i+1}, there exists an adversary for Game 1 which has non-negligible advantage.*

Fix i. Then as before we play Game 1.

R_i

(1) During re-randomization for CP_h, for $k \leq i$, replace the ciphertext in bin k with a random encryption of k.
(2) During decryption for CP_h, for $i \leq k$, calculate the expected decryption using the correct values and re-randomization factors $q_1 ... q_m$ and decrypt these on behalf of CP_h.

Figure 23: R_i, hybrids where rerandomization-reencryption is simulated

Our Re-encryption Re-randomization ZKP allows for an adversary to select $q = 0$ and present a proof that verifies correctly. But if Z selects $q = 0$ for any q, it must broadcast a pair of the form:

$$(g^{r0}, g^{(xr+m)0}) = (g^0, g^0) = (1, 1)$$

and that if this is observed by any honest CP, that CP executes ABORT. We present y_1 from Game 1 as the public key of CP_h, and set y_2 as the product of the public keys for the other CPs.

During re-randomization for CP_h, for bins k where $k < i + 1$, present a random encryption of k. For bin $i + 1$, find the plaintext p_{i+1} in bin $i + 1$, select a uniformly random $q_h \neq 0$ and set m_0 in Game 1 as $p_{i+1}q_h$. Set m_1 as $i + 1$, and present α, β from Game 1 as bin $i + 1$. Then for $k > i + 1$, present a correct reencryption-rerandomization of the ciphertext in bin k.

If $b = 0$ in Game 1, then the value in bin $i + 1$ is:

$$(g^{r'}, g^{r'x+p_{i+1}q_h})$$

for r' uniformly random. We claim this is a uniformly randomly selected re-encryption re-randomization of the previous value $(g^r, g^{rx+p_{i+1}})$.

This means it has the form:

$$(g^{(r+s)q}, g^{(r+s)qx+qp_{i+1}})$$

for random $s, q \neq 0$. For random $r', q_h \neq 0$ we show that there exist unique s, q which satisfy this form.

Set $q = q_h$, then set $r' = (r + s)q$ so that $s = r'q^{-1} - r$. Then it is easy to see the ciphertext

$$(g^{r'}, g^{r'x+p_{i+1}q_h}) = (g^{(r+s)q}, g^{(r+s)qx+qp_{i+1}})$$

which is a valid re-encryption re-randomization using parameters q, s.

Since $q = q_h$ they are selected identically, and we consider s. It is easy to see each s defines a unique r' by the linear mapping defined above and so we conclude for each pair (q, r') with $q \neq 0$, there is a unique pair (q_h, s) with $q_h \neq 0$ so we conclude (α, β) in this case is a uniformly random re-encryption re-randomization of the previous value if $b = 0$ so this is R_i.

Then if $b = 1$, (α, β) is an encryption of $i + 1$ exactly as in R_{i+1}.

We respond to the challenger in Game 1 with the output of Z and conclude the advantage of Z in distinguishing between R_i, R_{i+1} is equal to the advantage in Game 1 defined above, and so must be negligible.

B.2.9 Re-randomization 2. We define the hybrid which selects uniformly random elements to replace each nonzero plaintext during decryption independent from the view of Z. More specifically, R^2_{b+n} is defined exactly as R_{b+n} except as follows:

R^2_{b+n}

(1) During decryption for CP_h, for every element k in the vector presented to CP_h, if the plaintext of element k is not zero, generate a random new nonzero plaintext p_k. If the plaintext of element k is zero, set $p_k = 0$. Present an encryption of the p_k to the remaining *CPs*.

Figure 24: R^2, reencryption-rerandomization view change hybrids

Lemma B.10. *R_{b+n} and R^2_{b+n} are identically distributed.*

In R_{b+n}, the re-randomization factor for CP_h, q_h, is selected during decryption completely independent of the view of Z up until that point. For every k in the final vector, note that if the plaintext in k is 0, R_{b+n} and R^2_{b+n} are identical. Otherwise, we compare selecting the final plaintext p_k randomly to selecting it based on a correct re-randomization. We know each re-randomization factor q_i is not zero: honest *CPs* never select zero, and if a dishonest *CP* presents a re-randomization with 0, all the honest *CPs* abort. So the final plaintext in this case is $q_1 q_h q_2$ with q_1, q_2 selected by Z, possibly 1 if $h = 1$ or $h = m$. Then for any nonzero randomly selected plaintext p_k there is a unique nonzero q_h so that $q_1 q_h q_2 = p_k$ which is easy to see that for every p_k, there is a unique $q_h = (q_1 q_2)^{-1} p_k$ so that a random selection of is identical to a random selection of the other and the final plaintexts in each hybrid are identically distributed, which is the only difference between R_{b+n}, R^2_{b+n}

B.2.10 Ideal Model. All of the behavior in R^2_{b+n} is exactly what is done by the simulator except S does not have access to the increments given to the honest *DPs* and instead must get this value combined with the noise from $\mathcal{F}_{PSC}$. Recall that the output of $\mathcal{F}_{PSC}$ is the componentwise OR of the increments given by the honest *DPs*, and $\vec{v}^s$ which is the plaintexts submitted by corrupt *DPs* and *CPs* as extracted from the vector of submissions $\vec{c}$. However when an honest *DP* increments a bin in the ideal execution, $\mathcal{F}_{PSC}$ records this bin as nonzero regardless of any other behavior while in the hybrid R^2_{b+n} it is possible for honest *DPs* to increment some bin and then have this reversed by corrupt *CPs* and *DPs*, so that the bin is 0 in the R^2_{b+n} and nonzero in the ideal execution. We show this happens with negligible probability.

We define I, the final hybrid. More specifically, I is defined exactly as R^2_{b+n} except as follows:

Lemma B.11. *I outputs FAIL with negligible probability*

By construction, h_k is the inputs given to CP_h by the *DPs* along with a value v generated by CP_h. But no information about v is ever revealed or used until the decryption phase, which occurs after I chooses whether or not to output FAIL. Then for I to output FAIL, Z must guess a value v from which its view is statistically independent, so this happens with probability $\frac{1}{|\mathbb{Z}_q|}$ which is negligible.

I

(1) Record externally every increment instruction given to an honest *DP*. If some honest *DP* receives an increment instruction for some bin k, and during data submission the sum for that bin combined with h_k is 0, output FAIL.

Figure 25: I, a hybrid where data submission and aggregation is perfect

Lemma B.12. *The execution of I conditioned on not outputting FAIL is distributed identically to the view of Z interacting with S in the ideal model.*

The number of nonzero bins calculated by $\mathcal{F}_{PSC}$ is now calculated identically to the number in I, since no incremented bin saved by $\mathcal{F}_{PSC}$ by some honest *DP* is later reduced to 0 (this triggers a FAIL output). So this is identically the execution with S in the ideal model.

Theorem B.13. *The PSC protocol defined in section 4 UC-realizes the ideal functionality $\mathcal{F}_{PSC}$ in the ($\mathcal{F}_{ZKP-RR}$, $\mathcal{F}_{ZKP-DL}$, $\mathcal{F}_{ZKP-DLE}$, $\mathcal{F}_{ZKP-S}$, $\mathcal{F}_{BC}$)-hybrid model with point to point secure communication if at least one CP is honest and no CP is corrupted adaptively.*

I is a hybrid which is, conditioned on a negligible event FAIL not occurring, identical to S interacting with Z in the ideal execution, and D_0 is the ($\mathcal{F}_{RR}$, $\mathcal{F}_{DL}$, $\mathcal{F}_{DLE}$, $\mathcal{F}_{BC}$, $\mathcal{F}_{RS}$)-hybrid execution and we have a finite sequence of hybrids $D_0, ..., I$ which are all adjacentwise either identically distributed or computationally indistinguishable.