ABSTRACT

Many applications seek to produce differentially private statistics on sensitive data. Traditional approaches in the centralised model rely on a trusted aggregator to gather the raw data, aggregate statistics and introduce appropriate noise. Recent work has tried to relax the trust assumptions and reduce the need for trusted entities. However, such systems can trade off trust for increased noise and still require complete trust in some participants. Moreover, they do not prevent a malicious entity from introducing adversarial noise to skew the result or unmask some inputs.

In this paper, we introduce the notion of “verifiable differential privacy with covert security”. The purpose is to ensure both privacy of the client’s data and assurance that the output is not subject to any form of adversarial manipulation. The result is that everyone is assured that the noise used for differential privacy has been generated correctly, but no one can determine what the noise was. In the event of a malicious entity attempting to pervert the protocol, their actions will be detected with a constant probability negligibly close to one. We show that such verifiable privacy is practical and can be implemented at scale.

Keywords differential privacy, secure multiparty computation, covert security

1 Introduction

Efficient algorithms for computing accurate aggregate statistics over a population is one of the most fundamental and seemingly straightforward data analysis tasks with many applications. Fitness companies like Strava continuously monitor user activity to recommend popular running routes to their users [1]. Entertainment companies like Netflix and Hulu collect customer information to better recommend movies to their users [2,3]. Mapping services like Google Maps and Waze monitor user travel patterns to optimize route finding and traffic navigation [4]. Even the U.S Census Bureau publishes billions of aggregate statistics over its nation’s population and considers doing so a moral duty to maintain transparency [5]. Each of these scenarios requires the collection of reports from individuals; and the publication of aggregate statistics, which is represented via a histogram or set of (key, value) pairs.

The widespread nature of these applications and the size of the organisations providing them indicate a tremendous demand for such services. However, these benefits can come at the cost of privacy for individual participants: users disclose information about themselves which can be very sensitive. This leads to the risk of attackers being able to steal sensitive information from servers or cloud services [6], and misuse data for profit [7]. There are also scenarios when intentionally published data can be misused. For example, published census data has been used to discriminate against groups in society based on race [5]. Garfinkel et al. and Kasiviswanathan et al. describe practical reconstruction attacks that can be used to infer an individual’s private data from aggregate population statistics [8,9]. Thus, we seek ways to achieve both privacy and utility together without compromising on either. To achieve this middle ground, Dwork et al. proposed differential privacy, a principled framework for releasing aggregate statistics while respecting an individual’s privacy [10]. Differential privacy promises users that they will not be adversely or otherwise affected by allowing their data to be used in any study or analysis. Since its proposal, differential privacy has been adopted in government [11–13] and industry [14,15] and has become a de facto standard for releasing private data.

However, adopting differential privacy typically requires some level of explicit trust in one or more entities. In its most basic form, a centralized entity gathers reports from each individual data owner and adds appropriately scaled random
noise to the result of some aggregation. This aggregator must be trusted to perform the noise addition faithfully and not to allow the un-noised, unaggregated inputs to leak. Various relaxations of this model have been proposed to reduce the level of trust needed, but none are fully satisfying. Under distributed noise generation, the job of generating the noise is divided among multiple entities. However, if they are negligent in their duties, then un-noised results could be revealed. Alternately, in the local model (LDP) each client adds sufficient noise to mask their contribution. This has the disadvantage of introducing a much higher overall volume of noise with reduced utility. Instead, we would like to introduce only the minimum amount of noise, but with strong guarantees that either noise will be added correctly or those at fault will be exposed.

In this paper, we introduce and formalize this notion of “verifiable differential privacy with covert security”. We study this model applied to the foundational problem of estimating histograms over user data. Here, each user has a value \( x \) from a very large domain \( D \) and we want to publish a differentially private histogram over the population. Over the last decade, this problem has been studied almost ad nauseam \([10, 17–19]\). Algorithms are known that are optimal in terms of expected error, and which scale to millions of real-world users while preserving an individual’s privacy. Private histograms in turn enable a variety of other data analyses, such as heavy hitters and range queries \([20]\). However, all known efficient algorithms assume the semi-honest trust model \([21]\). Under semi-honest security, it is assumed that neither the central server(s) that collects information nor the individual clients deviate from the prescribed data collection protocol. This restriction makes semi-honest security unsuitable for overarching real-world deployment in the distributed setting. It is impractical to mandate millions of clients to behave exactly how we want, and we would like strong assurances that servers are not susceptible to leakage, or adversarial operation. Cheu, Smith and Ullman show how a few clients can deviate from the protocol to jeopardise the utility of LDP protocols completely \([22]\).

To prevent the central server from seeing a user’s inputs in plain text, the user may encrypt their inputs using cryptographic procedures like secret sharing \([23]\) and require multiple servers to compute aggregates over the shares cooperatively. Encryption protects honest users from the server but enables a malicious client to, for example, use a vast integer as input and trash the utility of the statistic. Accordingly, researchers have recently developed lightweight techniques to enable secret sharing to compute histograms even in the presence of malicious clients. PRIO \([24]\) (deployed at scale at Mozilla) and Poplar \([25]\) are systems where each user interacts with a small number of servers by sending them secret shares of their input. PRIO introduced SNIPS—a single prover, many honest verifier zero-knowledge proof system to deal with malicious clients. In Poplar, the authors similarly leverage lightweight sketching techniques \([26]\) which are secure even if a proper subset of the servers is malicious.

However, PRIO and Poplar require all servers to be semi-honest to guarantee correctness. If even one of the servers is corrupted by an active adversary, the honest parties cannot distinguish between an honest run and a corrupted run of the protocol. Consider the example of a plurality election with two servers and \( n \) voters. A corrupted server might not be interested in any user’s vote but in biasing the aggregate output of the protocol instead. Thus, if that server has auxiliary information about the preferences of a subset of users, the adversary might tamper with the protocol to exclude those honest voters from the election. In the case of differentially private outputs, the corrupted server could add arbitrary noise to the output, thereby eliminating any possible utility of the protocol. Furthermore, the honest server or the excluded voters cannot detect such an action. In another form of attack, a malicious voter could collude with the corrupted server to participate in ballot stuffing (skewing the result towards a particular outcome). PRIO and Poplar were not designed with this kind of attack in mind. In PRIO and Poplar, there is no verification system for each server to ensure that the other servers are behaving as expected. Under our security assumptions, full robustness against such attacks, where all attacks are detected and thwarted without revealing any private information, seems too strong a guarantee to hope for. The feasibility results of \([27]\) show that we cannot be robust to active adversaries if more than one third the participants deviate from protocol specifications. Like Prio and Poplar, we only require that one participant follow the protocol. Therefore, we operate under the practical assumptions of covert security \([28]\) instead. Under covert security, participants can still deviate arbitrarily and potentially violate an honest party’s privacy. However, they do so at the risk of getting caught with at least a constant probability greater than \( \frac{1}{3} \). In our protocols, we treat privacy and utility with equal importance. A corrupt party is caught with probability one if they deviate from the prescribed protocol in any way, even if the deviation did not violate user privacy. Thus at the cost of a modicum of efficiency, the users of our system benefit from the guarantee that as long as one server is semi-honest and does not abort, the protocol’s output is guaranteed to be correct.

### 1.1 Our Contributions

Given a population of \( n \) clients, each of whom holds a private input in \( [M] \), we design a protocol to output differentially private histograms. The following properties distinguish our protocol from prior work described below:

1. **Verifiable Robustness Of Output**: As in PRIO, clients distribute secret shares of their input among \( K \geq 2 \) servers. As long as a single server is semi-honest, an adversary controlling a client and any proper subset
of servers cannot bias the outcome of the protocol by including illegal inputs, excluding valid inputs from the protocol or tampering with the messaging specifications of the protocol, without being caught out by the honest server. In PRIO [24], Poplar [29], STAR [30] or sparse histogram estimation [31] these correctness properties hold only if the servers are semi-honest. Furthermore, the honest party is unable to detect deviations from the protocol.

2. Verifiable Differential Privacy: We show that as long as a single server is semi-honest, our protocol is $(\epsilon, \delta)$ differentially private. In PRIO, Poplar and sparse histogram estimation with homomorphic encryption, a server corrupted by an active adversary could tamper with the output, still satisfying differential privacy but compromising the protocol’s utility. On the other hand, we guarantee differential privacy while preserving utility guarantees. STAR does not address DP statistics.

Although built using familiar primitives (secret shares, commitments, etc.), our protocol combines these in a novel way. For example, we generate values that are guaranteed to be picked uniformly at random to ensure that the input can be verified as valid. Then, to ensure that the privacy noise is generated from the correct distribution, we build on this technique by expressing Binomial noise as a sum of Bernoulli random variables that can be generated from a mixture of public and private randomness, and summed using arithmetic shares.

After reviewing prior work in Section 2 and stating definitions in Section 3, we start by introducing a scheme that achieves privacy in the semi-honest model by adding discrete noise to shares of the input (Section 4). However, this scheme is not robust against a variety of adversarial actions from clients and servers. To remedy this, in Section 5 we show how the “tallying” (aggregation) part of the protocol can be performed using commitments to prevent dishonest actions of clients and servers. Then in Section 6 we show how the servers can introduce privacy noise such that the noise follows the correct distribution. However, no individual server can learn the overall noise used to achieve differential privacy, but any server can verify if another server used inappropriate noise despite not knowing the noise values. Finally, we provide proof-of-concept evaluations to compare the computational costs of verifiable vs. unverifiable privacy in Section 7

All protocols in this paper are secure against covert adversaries. Aumann and Lindell introduced covert adversaries as a compromise between active and passive adversaries [28]. Passive adversaries cannot force MPC participants to deviate from the specified protocols. On the other hand, active participants are not required to follow protocol specifications. This implies that participants might use a different input from the one specified on their input tape and, at any point in the protocol, deviate from the protocol and send arbitrary messages. In covert security, the adversary has the same agency as the adversaries in active security—they may arbitrarily deviate from the prescribed protocol. Doing so could violate an honest participant’s privacy, but such a deviation also risks reputational harm to the adversary. A covertly secure protocol (under the explicit cheat model) allows the honest parties to detect adversarial behaviour with a constant probability $\frac{1}{2} < \rho \leq 1$.

1.2 Notation

We write $x \overset{R}{\leftarrow} U$ to denote that $x$ was uniformly sampled from a set $U$. Let $p$ and $q$ denote appropriately selected large primes such that $q$ divides $p - 1$ and $q$ is much larger than the number of clients $n$. Let $\mathbb{G}_q$ be a unique subgroup of $\mathbb{Z}_p^*$ with order $q$. Let $g$ denote a randomly selected generator for group $\mathbb{G}_q$ and $h = g^\alpha$, for some $\alpha \overset{R}{\leftarrow} \mathbb{Z}_q$. We assume that the discrete log (DL) problem defined in Section 5 is intractable in $\mathbb{G}_q$ for any polynomial-time algorithm $\mathcal{A}$. For any integer $M$, let $[M]$ denote the set $\{1, 2, \ldots, M\}$. Throughout this paper, when we write $a + b \in \mathbb{Z}_q$, we mean $a + b \text{ mod } q$. We will denote vectors with boldface font and an arrow on top as in $\vec{v} \in \mathbb{Z}_q^M$. Throughout the paper, we will represent a client input encoded as a 1-hot vector in $\mathbb{Z}_q^M$. A client input is legal if it is non-zero at a single coordinate in $\mathbb{Z}_q^M$ and $\|\vec{v}\|_1 = 1$. That is, the set of legal inputs to a $M$ bucket histogram protocol is given by

$$V = \{\vec{v} \in \mathbb{Z}_q^M : v_i = 1 \text{ and } v_j = 0 \text{ for all } j \neq i\}$$

(1)

2 Related Work

Differential Privacy. In one of the earliest works on differential privacy, Dwork et al. described the Laplace mechanism for outputting histograms under pure differential privacy in the trusted curator model with passive security [10]. Soon after, McSherry et al. proposed the exponential mechanism [32] or equivalently the report noisy max [33], which allows us to compute the (approximately) most frequent bucket in a histogram, also under pure differential privacy. Although these mechanisms give us pure differential privacy and optimal error rates $O(\frac{1}{\epsilon})$, implementing such a “central” model requires trusting that the curator would follow the protocol and not adversarially use client data that it sees in plaintext.
Verifiable Differential Privacy For When The Curious Become Dishonest

Building histograms over distributed data can be viewed as a multiparty computation. Secure multiparty computation (MPC) allows $K$ participants to compute a function over each other’s inputs without revealing any information about them. Recently there has been an influx of research using MPC protocols to release private aggregate statistics. Fig. 1 summarises the latest MPC protocols that have been used to compute DP statistics. Researchers at Brave use oblivious pseudorandom functions \cite{40} to compute succinct histograms of client inputs in the two server setting \cite{50}. Researchers at Google use homomorphic encryption to compute differentially

| Trust         | Robust | Comm  | DP-Error |
|---------------|--------|-------|----------|
| Sparse Histograms | Semi-honest | No     | $O(\ln M)$ | $O(1)$ |
| Prio          | Semi-honest | No     | $O(M)$   | $O(1)$ |
| Brave STAR    | Semi-honest | No     | $O(\ln M)$ | No-DP    |
| Poplar        | Active  | No     | $O(\ln M)$ | $O(\ln M)$ |
| Our Work      | Covert $\rho = 1$ | Yes    | $O(M)$   | $O(1)$ |

Figure 1: The table above compares recent results in MPC for DP. The first column refers to the trust assumptions placed on the servers. Note that Sparse Histograms have additional DP leakage and STAR can only guarantees $K$-anonymity. Robustness refers to the guarantees of utility if the protocol was not aborted. The last two columns compare the error introduced by DP noise and the client-sever communication complexity. It should be noted that the DP-error in PRIO, Poplar and our work has a linear dependence on the number of servers, $K$, which is assumed to be small ($K \leq 6$). Hence we do not include it.

Therefore, researchers studied local privacy (LDP) \cite{34} using randomised response \cite{35} to prevent any other party from seeing data in plaintext. Recently, Cheu, Smith and Ullman showed that the randomised response algorithm generalises all locally private protocols \cite{22}. This generalisation highlights two unavoidable disadvantages of local differential privacy. The first is that the accuracy of the protocol for even the binary histogram is $O(\sqrt{n})$ compared to $O(1)$ in the central model. The second is that randomised response systems offer a much weaker definition of privacy than the usual cryptography standards such as semantic security. For example, if the client flips their original answer with probability $p = 0.1$, the curator sees their sensitive information in plain text 90% of the time. Further increasing $p$ reduces the accuracy of the protocol dramatically. Consider the example described in \cite{54}, where 1% of the population of a million people answer “yes” to a survey about a sensitive topic. If we set $p = 0.49$, then one-third of the time, the central analyser concludes that not a single person member of the population answered “yes”. Thus if we wanted to preserve correctness, this definition of security is considerably weaker than the indistinguishability guarantees provided by protocols such as secret sharing.

Shuffle privacy \cite{19,36,37} analyses local mechanisms under the lens of central privacy and bridges the accuracy gap between local and central models. Recent results \cite{18,38} prove that near central error guarantees are possible with distributed local transformations. Although this bypasses the accuracy issue of LDP, shuffle privacy assumes the existence of a secure shuffler, which is non-trivial to implement. In recent work, Bell et al. show that secure aggregation realises secure shuffling \cite{39}. However, such protocols impose the impractical constraint of secure peer-to-peer communication between clients, and the curator is still a single source of failure. Despite the immense progress on differentially private histogram estimation, all known efficient implementations assume semi-honest participants and are a variant of either randomised response or the additive mechanism. It only takes a small fraction of clients to deviate from their prescribed protocol to destroy any utility of randomised response \cite{22}. Additive mechanisms involve adding carefully curated randomness to the statistic before being released as output. It is easy to see how Byzantine participants could violate the utility and privacy of additive mechanisms. For example, in the case of central privacy, the trusted server could simply output any value of their choice instead of following the protocol. Or, the central server could reveal the randomness to an adversary $\mathcal{A}$, who is then able to subtract randomness from the output. The crux of the problem is that the participating clients do not have the means to validate the server’s actions while receiving low error private aggregate statistics.

Secure Multi Party Computation. Building histograms over distributed data can be viewed as a multiparty computation. Secure multiparty computation (MPC) allows $K$ participants to compute a function over each other’s inputs without revealing any information about them. Recently there has been an influx of research using MPC protocols to release private aggregate statistics. Fig. 1 summarises the latest MPC protocols that have been used to compute DP statistics. Researchers at Brave use oblivious pseudorandom functions \cite{40} to compute succinct histograms of client inputs in the two server setting \cite{50}. Researchers at Google use homomorphic encryption to compute differentially
private sparse histograms in two-server models (2PC) \[31\]. Gibbs and Boneh propose PRIO, an MPC protocol between a small number of servers that receive shares of client input instead of the plaintext input to compute differentially private histograms \[24\]. The methods described above alleviate the need for the central server to see inputs in plaintext form. Sketching techniques introduced by Boyle et al. can be used in conjunction with PRIO to ensure malicious clients cannot tamper with the protocol output by sending invalid inputs \[26\]. For high dimensional inputs, Boneh et al. use distributed point functions (DPFs) \[41\] in the two server model to reduce communication overhead by a log factor \[29\]. Although these methods have led to efficient MPC protocols that operate at scale and can deal with malicious clients, they still require that the servers remain semi-honest and non-colluding. An active adversary that is not bound to follow the specifications of the protocol could violate the privacy and utility of these protocols and go undetected\[^{[1]}\]

3 Preliminaries

Throughout the paper, we assume that the clients can communicate with individual curators using secure encrypted channels. Clients are also considered to have access to a broadcast channel to communicate with all curators. The curators are also assumed to have secure encrypted channels to talk with other curators. This can be accomplished using a secure, authenticated public key infrastructure. We make no assumptions about the synchronicity of the network. An adversary may arbitrarily drop or re-order packets over the network. Additionally, an adversary is allowed to monitor all communications over the wire.

Covert Secure Computation.

Definition 1. Let \( f \), \( \pi \), and \( p \) represent the ideal functionality, real-world protocol and deterrent factor. Protocol \( \pi \) is said to securely compute \( f \) with abort in the presence of covert adversary with \( \rho \)-deterrent, if for every non-uniform \( A \) in the real model, there exists a non-uniform PPT adversary \( S \) in the ideal model, such that for a corrupted party \( P_1 \), we have

\[
\text{IDEAL}_f,S(\pi,n) = \text{REAL}_{\pi,A}(z,\bar{x},n) = \text{IDEAL}_{f,S(\pi),P_1}(\bar{x},n) \quad \forall n \in \mathbb{N}
\]

where, \( \bar{x} \) is the input used by \( K \) parties; \( z \) is the auxiliary input available to \( S \) and \( A \); and \( n \) is the security parameter (which, for all protocols discussed in the paper, is identical to the length of the inputs). \( \text{REAL}_{\pi,A}(z,\bar{x},n) \) refers to the joint distribution of \( A \)'s view and output in the real world and \( \text{IDEAL}_{f,S(\pi),f}(\bar{x},n) \) refers to the joint distribution of the output and view of \( S \) in the ideal world.

Modular Sequential Composition.

Theorem 3.1. \[^{[28]}\] Let \( p(n) \) be a polynomial, and let \( f_1, \ldots, f_p(n) \) be a multiparty PPT functionality. Let \( \pi_1, \ldots, \pi_p(n) \) be protocols that securely compute \( f_1, \ldots, f_p(n) \) in the presence of covert adversaries with deterrent \( \rho_1, \ldots, \rho_p(n) \) respectively. Let \( g \) be a multiparty functionality and let \( \Pi \) be a secure protocol for computing \( g \) in the \( \langle f_1, \rho_1 \rangle, \ldots, \langle f_p(n), \rho_p(n) \rangle \) hybrid model (using a single call to each \( f_i \) in presence of covert adversaries). Then \( \Pi^{\pi_1, \ldots, \pi_p(n)} \) securely computes \( g \) in presence of the same covert adversaries in the real world.

In this work, we make extensive use of modular composition. We first define protocols that compute more straightforward multiparty functionalities securely. We then describe the more complicated protocols in a hybrid world and use modular sequential composition to prove the security of the final protocol.

Differential Privacy.

Definition 2. An algorithm \( M : X^n \times Q \rightarrow Z \) satisfies \((\epsilon, \delta)\) differential privacy if for every two neighboring datasets \( X \sim X' \) and for every query \( q \in Q \) we have for all \( T \subseteq Z \)

\[
\Pr[M(x, q) \in T] \leq e^{\epsilon} \Pr[M(x', q) \in T] + \delta
\]

Pedersen Commitments. Let \( g \) be a generator of \( \mathbb{G}_q \) and \( h \) be a random element of \( \mathbb{G}_q \). For \( r \leftarrow \mathbb{Z}_q \), a Pedersen commitment to \( s \in \mathbb{Z}_q \) is defined as

\[
\text{Com}(s, r) = g^s h^r = c
\]

The commitment has the following properties:

Theorem 3.2. \[^{[42]}\] For any \( s \in \mathbb{Z}_q \) and \( r \leftarrow \mathbb{Z}_q \), \( \text{Com}(s, r) \) is uniformly distributed in \( \mathbb{G}_q \). Therefore the commitment is information-theoretically hiding.

\[^{[1]}\] Poplar \[^{[29]}\] is private against corrupt servers but does not guarantee correctness
Verifiable Differential Privacy For When The Curious Become Dishonest

We will also highlight where malicious servers would break the protocol, setting the stage to extend it to achieve verifiability. The protocol consists of three phases, namely, (a) Voting, (b) Verification and (c) Tallying. When the servers are semi-honest, one can be trusted to add correctly calibrated smooth noise to the protocol’s output. Given the semi-honest nature of the curators, we can trust that the noise remains secret.

Figure 2: An overview of protocol for verifiable histogram estimation. Values sent to the billboard represent broadcasted messages that are public knowledge. The protocol consists of three phases, namely, (a) Voting, (b) Verification and (c) Tallying. When the servers are semi-honest, one can be trusted to add correctly calibrated smooth noise to the protocol’s output. Given the semi-honest nature of the curators, we can trust that the noise remains secret.

Theorem 3.3. \[42\] Any adversary \( \mathcal{A} \) that can break Pedersen commitments can also win the DLog game with non-negligible probability. More formally, given \( \text{Com}(s, r) \) if \( \mathcal{A} \) can find \( (s', r') \) such that \( \text{Com}(s, r) = \text{Com}(s', r') \) with probability \( \rho \), then \( \mathcal{A} \) is able to win the DLog game with probability \( \rho \). Thus the commitment is computationally bounding.

Notation: For a vector \( \vec{s} \in \mathbb{Z}_q^M \) and i.i.d random values \( r \sim \mathbb{Z}_q^M \), when we write \( \text{Com}(\vec{s}, r) \in G_q^M \) we mean coordinatewise commitments of \( \vec{s} \) using \( r \) i.e., for each \( j \in [M] \), \( \text{Com}(s_j, r_j) = g^{s_j} g^{q r_j} \).

Morra. Define \( f_{\text{morra}}(\lambda_1, \ldots, \lambda_K) = z \) where \( z \sim \mathbb{Z}_q \) and \( \lambda_k \) refers to the empty string for all \( k \in [K] \). The following protocol \( \pi_{\text{morra}} \) shown in Algorithm 1, which is a modification of an ancient game called Morra, securely computes \( f_{\text{morra}} \) in the presence of active adversaries. It is easy to see that as long as one server is honest and samples their value uniformly at random, \( z \sim \mathbb{Z}_q \). Additionally, since the Pedersen commitment is unconditionally hiding, a corrupt server cannot infer any information about a server’s choice of \( m_k \) from the published \( c_k \). A formal simulator-styled proof can be found in Blum’s seminal work for flipping coins over a telephone \[43\]. If we omit the final thresholding step, the above protocol can be used to sample \( z \sim \mathbb{Z}_q \).

Algorithm 1 \( \pi_{\text{morra}} \) A protocol for sampling a public coin

**Input:** \( \lambda_1, \ldots, \lambda_K \)

**Output:** \( z \sim \mathbb{Z}_q \)

1. Each server \( k \in [K] \) samples \( m_k \sim \mathbb{Z}_q \) uniformly at random.
2. **Commit:** Each server samples \( r_k \sim \mathbb{Z}_q \) and broadcasts \( c_k = \text{Com}(m_k, r_k) \) to all other servers.
3. **Reveal:** Once all servers have received \( c_k \), they now broadcast \( m_k, r_k \) to all servers. Each server verifies that \( \text{Com}(m_k, r_k) = c_k \). If this test fails for any \( k \) or one of the servers does not respond, the protocol is aborted.
4. Each server computes \( X = (m_1 + \cdots + m_k) \mod q \). We have \( X \sim \mathbb{Z}_q \). If \( X \leq \lceil \frac{q}{2} \rceil \) then \( c_i = 0 \). Otherwise \( c_i = 1 \). Thus we can use this protocol to generate unbiased coins and uniformly random values.

4 Warm-up: DP histograms with semi-honest curators

We start by building a protocol for the exact problem with semi-honest curators and byzantine clients, and then extend it to provide differential privacy. This mimics the Poplar and PRIO systems.

We will also highlight where malicious servers would break the protocol, setting the stage to extend it to achieve verifiability. The protocol proceeds in the three phases (illustrated in Figure 2) (1) voting—the clients create additive shares of their input, (2) verification—the servers check if the client’s input is valid and (3) tallying—the servers aggregate their shares to output a histogram.

\[2\] When the curators are semi-honest, there is no need for public commitments.
Algorithm 2 Semi-Honest Voting Phase

**Input:** Client input $\vec{v} \in \mathbb{Z}_q^M$

**Output:** Shares $[\vec{v}]_1, \ldots, [\vec{v}]_K$

1. The client samples $[\vec{v}]_j \leftarrow \mathbb{Z}_q^M$ for $j = [K-1]$
2. The client sets $[\vec{v}]_K = \vec{v} - \sum_{k=1}^{K-1} [\vec{v}]_k$. It then sends one share to each server.

Algorithm 3 Semi-Honest Verification Phase

**Input:** Shares $[\vec{v}]_1, \ldots, [\vec{v}]_K$

**Output:** Decision $f_{\text{verify}}(v) = I(v \in V)$

1. One of the servers samples $\vec{r} = (r_1, \ldots, r_M) \leftarrow \mathbb{Z}_q^M$. It then computes $\vec{r}^* = (r_1^2, \ldots, r_K^2)$ and broadcasts $\vec{r}$ and $\vec{r}^*$ to the other servers. Each server $k \in [K]$ computes the inner product and broadcasts $z_k$ to all other servers, as shown below

$$z_k = \langle \vec{r}, [\vec{v}]_k \rangle \in \mathbb{Z}_q$$
$$z_k^* = \langle \vec{r}^*, [\vec{v}]_k \rangle \in \mathbb{Z}_q$$  \hspace{1cm} (5)

2. **Decision:** Each server independently computes $z = z_0 + \cdots + z_k$ and $z^* = z_0^* + \cdots + z_k^*$. The servers accept input $v \iff (z^2 - z^*) = 0$  \hspace{1cm} (6)

4.1 Handling byzantine clients

In the voting phase described by Algorithm 2, an honest client splits their input into $K$ additive shares, which implies that unless all servers colluded with each other, the client’s input is perfectly private (Shannon Security). The histogram estimation protocol is a linear aggregation over client inputs, so that each server can apply aggregations on their shares instead. It is unnecessary for any server to see the plaintext input at any point in the protocol. By linearity of secret sharing, we will recover the desired output when we reconstruct using the individual share aggregations.

The only adversarial behaviour we must handle here is to prevent clients from submitting invalid inputs $v \notin V$. Boyle et al.’s sketching protocol [26] described in Algorithm 3 ensures that clients cannot submit illegal inputs and any servers corrupted by a passive adversary do not learn anything about the client’s inputs. Theorem 4.1 lists the formal guarantees of the verification protocol. We remind the reader that for semi-honest servers, Algorithm 3 is not novel and in fact borrowed exactly from [26].

**Theorem 4.1.** Assuming that all $K \geq 2$ servers are semi-honest, the protocol described in Algorithm 3 satisfies the following properties:

- **Completeness:** If a client’s input $\vec{v} \in V$, then the servers always admit the input into the protocol.

- **Soundness:** If the vote is illegal i.e., $\vec{v} \notin V$, then for all corrupt clients (even those that are computationally unbounded), the servers admit the input with cryptographically small probability $\frac{1}{q}$. This probability can be made arbitrarily small by repeating the protocol in parallel.

- **Zero-Knowledge:** The servers learn nothing more than the information that the client input is valid. More precisely, when $\vec{v} \in V$, each party can simulate its view of the protocol without ever talking to the proving party.

A corresponding theorem is shown in [26]; for completeness, we provide a proof in Appendix A.

**Challenges for handling active adversaries.** Before proceeding to differentially private outputs, we point out how the protocol breaks down in the presence of active adversaries. In the presence of even one corrupt server, the semi-honest protocol described in Algorithm 3 always leaks at least one bit of information. Consider the following scenario where
Verifiable Differential Privacy For When The Curious Become Dishonest

Figure 3: Shown above are the two types of attacks that go undetected in Poplar. In (a) regardless of what the honest client sends, a corrupted server simply ignores the input and excludes the client from the protocol based on auxiliary information. In (b) dishonest client teams up with the corrupted server and lets the server know the value of κ and [v]₀. This allows the server to set z₁ = −z₀, z*₁ = −z*₀ and z**₁ = −z**₀, thereby allowing an illegal input into the protocol. In both cases, the honest server cannot distinguish between an honest run and a corrupted run of the verification protocol.

Algorithm 4 Semi-Honest Tallying Phase

**Input:** Additive shares of n honest clients  
**Output:** Differentially private histograms

1. The k’th server computes \( \vec{z}_k = \sum_{i=1}^{n}[\vec{v}]_k \) and broadcasts it to the other servers. Note \( \vec{z}_k \) is uniformly distributed over \( \mathbb{Z}_M^q \). So the other servers learn nothing about the client inputs from \( \vec{z}_k \).

2. One of the K servers has the designated role of a randomiser. Let \( j \) represent that server. The \( j \)’th server computes \( \vec{\eta} = (\eta_1, \ldots, \eta_M) \) where \( \eta_j \sim \text{Bin}(n_b, \frac{1}{2}) \). It broadcasts \( \vec{z}_j = \vec{\eta} + \sum_{i=1}^{n}[\vec{v}]_k \).

3. After each server is done broadcasting, they each aggregate broadcasted shares to compute \( \vec{z} = \sum_{i=1}^{n}[\vec{z}]_i \) and publishes it in the open. By the linearity of additive secret sharing, we get differential privacy.

\[
\vec{z} = \sum_{k=1}^{K} \vec{z}_k = \vec{\eta} + \sum_{i=1}^{n} \sum_{k=1}^{K} [\vec{v}]_k = \vec{\eta} + \sum_{i=1}^{n} \sum_{k=1}^{K} \vec{v}_i
\]  

(7)

A might guess that an honest client’s input is non-zero at the \( j \)’th index. The adversary has the corrupt server decrement the value of their share by one at index \( j \) and increments it at another index of their choosing (a “sliding attack”). They then run the protocol as prescribed before. If the test passes, \( A \) knows it successfully guessed the client’s vote. Ignoring any access to auxiliary information, this event happens with a probability of at least \( \frac{1}{M} \) (which is not negligible for small \( M \)). If the guess was incorrect, the test fails, but the adversary now knows that the client did not select \( j \).

In Poplar, the authors modify the sketching protocol above to protect an honest client by having the client randomly sample a secret \( \kappa \in \mathbb{Z}_q \) and send the servers, \([\vec{v}]_k\) and \([\vec{v}]_k^* = [\kappa \vec{v}]_k\). To pass the test, the adversary must guess \( \kappa \) and \( j \), and the probability of such an event is negligible in the size of the finite field. We refer the reader to Appendix C of [29] for details. Although this improvement removes the single bit of leakage, it does not protect the protocol from the following scenarios.

A corrupted server could completely ignore the contents of a client’s input and send arbitrary messages during the verification protocol. This results in the client being excluded from the protocol. For example, if the protocol was used for voting purposes, an adversary with side information about a client might choose to exclude them from participating in voting, thereby biasing the election. Furthermore, an adversary \( A \) controlling a single server can collude with a client to include an illegal input into the protocol, which leads to unreliable outputs once again. In Poplar, these attacks go undetected. Figure 3 summarises the attacks against Poplar. In our protocols, under the protection of covert security, it is acceptable to leak one bit of information or for an adversary to tamper with the output but only provided that the honest party can detect such behavior and abort with constant probability.

4.2 Ensuring differential privacy

In Poplar and PRIO, each server independently adds Laplacian noise to the aggregate of their shares to ensure that the output is differentially private. However, if an active adversary corrupts a server, we can no longer assume that
Verifiable Differential Privacy For When The Curious Become Dishonest

the additive noise is correctly calibrated. An adversary may bias the protocol’s output by tampering with the noise distribution. Instead, we will construct an MPC protocol for implementing the Binomial mechanism, where noise follows a Binomial distribution. The amount of noise from the Laplace and Binomial mechanisms is comparable, but due to the discrete nature of the noise, the Binomial distribution more conveniently lends itself to our subsequent proofs of verifiability and security.

Algorithm 4 adopts the binomial mechanism as described by Balcer and Cheu [38] in the semi-honest setting. Each server aggregates its shares and broadcasts the aggregate to other servers. By linearity of secret-sharing, we get exact histograms. A designated server is trusted with adding binomial noise to ensure DP. Theorem 4.2 re-states the DP guarantee provided by the mechanism.

In addition to trusting the servers to add noise correctly, Poplar and PRIO also assume that the servers will not tamper with the shares after they have been accepted. A corrupted server might include an honest client’s input but fiddle with the shares later, compromising the aggregation protocol’s utility.

Throughout the paper, when we say computationally bounded adversary \( \mathcal{A} \) to control clients and up to \( K - 1 \) servers.

This section and the next describe our primary contributions, where we allow a computationally bounded adversary \( \mathcal{A} \) to control clients and up to \( K - 1 \) servers. Throughout the paper, when we say computationally bounded adversary \( \mathcal{A} \), we mean that the advantage of \( \mathcal{A} \) in solving the discrete log problem described in Appendix D is negligible. Under covert security, it is acceptable for our protocol to leak some information in the presence of adversaries that may deviate arbitrarily. However, we require that honest parties detect any such deviation with constant probability greater than \( \frac{1}{2} \). The assumption behind covert security is that an adversary will not risk reputational harm and cheat if the likelihood of detection is significant. In our protocols, tampering with the messaging specifications may compromise a protocol’s privacy or correctness property. As discussed
Verifiable Differential Privacy For When The Curious Become Dishonest

Algorithm 5 Active voting

**Input:** Client input $\vec{v} \in \mathbb{Z}_q^M$

**Public Parameters:** $g, h \in \mathbb{G}_q$, $M$ and $\mathbb{Z}_q$

**Output:** Committed Shares $(\vec{v}_1, \alpha_1^1), \ldots, (\vec{v}_K, \alpha_K^K)$

1. Let $\vec{v} = (v_1, \ldots, v_M)$ denote a clients input. An honest client uses additive secret sharing to generate shares $(\vec{v}_1, \ldots, \vec{v}_K)$.

2. For each share $k \in [K]$ the client computes a Pedersen commitment $\text{Com}(v[k], t_k) = \alpha_k^r$ where $t_k \in \mathbb{Z}_q^M$, $\alpha_k = (\alpha_1, \ldots, \alpha_K)$. Let $[v_j]_k = \text{Index}(v[j], k)$ and $t_{jk} = \text{Index}(t_k, j)$, where $\text{Index}(x, p)$ is refers to the $p$'th coordinate of the vector $x$. Then $\alpha_{jk} = \text{Com}(v[j], t_{jk})$ for $j \in [M]$ as described in Section 3.

It broadcasts this commitments publicly for everyone to see. Thus for each client input $\vec{v}$, there is a total of $KM$ commitments in $\mathbb{G}_q$. Given that these commitments are information-theoretically hiding, to any adversary $A$ they are indistinguishable from $KM$ random values independently sampled from $\mathbb{G}_q$.

3. The client sends $[\vec{v}]_k$ and $t_k$ to each server $k$ over a secure point-to-point channel.

before, we give equal importance to privacy and utility. If we cannot guarantee that the output of a protocol is reliable, we cannot reliably use it for real-world tasks. Hence, we require that the honest party be able to detect any deviation from the prescribed protocol and not just violations of individual privacy. Our protocol proceeds in three phases: voting, verification and tallying, as described in Fig 5. In the voting phase, described in Algorithm 5 in addition to sending secret shares to servers, the clients are required to commit to the secret shares they send to servers publicly. The honest server can use these public commitments to detect if an adversary caused the servers to tamper with client shares. It can do so as the verification and tallying phases are linear, and the commitment schemes are homomorphic.

Once the clients have committed to their shares, the servers follow the same sketching test described in Algorithm 5 with an important extra step. The sketching test is preceded by an authentication test which ensures that the servers can be combined to describe a single curator indexed by 1. Likewise, we represent all corrupt curators with a protocol. A simulator proof is deferred to Appendix C.

**Theorem 5.1.** Let $\vec{v}$ represent an honest client’s input and $f_{\text{verify}}$ denote the deterministic function that checks if $v \in V$. Let $\pi_{\text{sketch}}$ refer to the protocol described in Algorithm 6. Then $\pi_{\text{sketch}}$ securely computes $f$ with abort in the presence of static covert adversary with $\rho$-deterrent, where $\rho = 1 - \text{Adv}_A(DLog) \approx 1$.

The proof for the above theorem can be found in Appendix C.

**Lemma 5.2.** If a static covert adversary $A$ controlling up to $K - 1$ servers deviates in any way from the messaging protocol $\pi_{\text{sketch}}$, it gets caught by the honest server with probability $1 - \text{Adv}_A(DLog) \approx 1$.

**Proof.** We will prove the above theorem for $K = 2$ as it generalises all values of $K \geq 2$. The output of all honest servers can be combined to describe a single curator indexed by 1. Likewise, we represent all corrupt curators with a single index 2. We start by assuming an honest client has committed and distributed shares of its input to both servers. Let $z_2, z_2^* \in \mathbb{Z}_q$ and $u_2, w_2 \in \mathbb{Z}_q$ be the messages Server 2 sends Server 1 had they not been corrupted by $A$ as specified by the protocol.

The only way for $A$ to deviate from $\pi_{\text{sketch}}$ is by sending $\hat{z}_2, \hat{z}_2^* \in \mathbb{Z}_q$ and $\hat{u}_2, \hat{w}_2 \in \mathbb{Z}_q$ such that at least one of the values are mismatched and is not what it was meant to be. To go undetected, $A$ must pass both the sketching and authentication tests. To pass the sketching test, it must have both $\hat{z}_2 \neq z_2$ and $\hat{z}_2^* \neq z_2^*$. Otherwise, we will not have $\hat{z}_2 - \hat{z}_2^* = 0$. Now assume that $A$ passes the authentication test with non-negligible probability. This would imply that
Algorithm 6 Covert Verification Phase

Input: For a client input \( \vec{v} \), each server \( k \in [K] \) receives committed shares \( \langle [\vec{v}]_k, [\vec{t}]_k \rangle \).

Honest server Output: The honest server either aborts (and outputs \( \perp \)) or \( f_{\text{verify}}(\vec{v}) = I(v \in V) \)

Public Parameters: \( g, h \in \mathbb{G}_q, M \) and \( \mathbb{Z}_q \), commitments to shares

1. For a given client input, the \( k \)-th server has \( t_k \) and \( [\vec{t}]_k \). The server checks if
   \[
   \text{Com}([\vec{v}]_k, [\vec{t}]_k) = \alpha_k \quad (10)
   \]
   If this test fails, it implies that the client did not commit to their shares correctly. Thus the clients inputs are excluded from the future steps of the protocol and this client plays no further role in the remainder of the protocol.

2. Random coin generation: The servers play Morra (as described in section 3) to generate public random vectors \( \vec{r} = (r_1, \ldots, r_M) \in \mathcal{R} \mathbb{Z}_q^M \). It is important to remember that the client has already committed to their shares before these random public coins are generated.

3. Local Computations at Server: Each server \( k \) computes dot products and broadcasts to all other servers
   \[
   z_k = \sum_{j=1}^{M} r_i [v_i]_k, z^*_k = \sum_{j=1}^{M} r_i^2 [v_i]_k, u_k = \sum_{j=1}^{M} r_i t_i \text{ and } w_k = \sum_{j=1}^{M} r_i^2 t_i
   \]

4. Authentication Test: For each \( k \in [K] \), every server (including the honest one) checks if the following tests pass. If any of the authentication tests fail for server \( k \), it implies that Server \( k \) deviated from the protocol specification. Thus, the honest server aborts the protocol and outputs \( \perp \). Otherwise, the servers proceed to the sketching test.
   \[
   \prod_{i=1}^{M} (\alpha_{ik})^{r_i} = g^{z_k} h^{u_k} \\
   \prod_{i=1}^{M} (\alpha_{ik})^{r_i^2} = g^{z^*_k} h^{w_k} \quad (11a, 11b)
   \]

5. Sketch test: Each server computes \( z = z_0 + \cdots + z_k \), \( z^* = z_0^* + \cdots + z_k^* \) and \( u = u_0 + \cdots + u_k \). They then check if the sketching test \( (12) \) passes. If this test fails, it implies that the client sent an illegal input. Therefore the servers excluded this client’s input from the protocol altogether. Otherwise, the honest server outputs
   \[
   (z^2 - z^*) = 0 \quad (12)
   \]

By the computationally binding property of Pedersen commitments given in Theorem 3.3, this would imply the adversary was able to break the discrete log game with non-negligible probability (which violates the static covert adversary assumptions used in this paper). Thus \( \mathcal{A} \) cannot exclude an honest client from the protocol without getting caught with probability \( 1 - \text{Adv}_{\mathcal{A}}(\text{DLog}) \approx 1 \).

An immediate result of the Lemma 5.2 is that it protects honest parties from collusion between clients and servers. This contrasts with the active secure sketching protocols in [25], where a client and a server can collude to include illegal inputs in the protocol.

**Corollary 5.2.1.** A static covert adversary \( \mathcal{A} \) that controls up to \( K - 1 \) servers and a malicious client cannot include an illegal input into the protocol without getting caught by the honest server, with success probability at most \( \max\{\text{Adv}_{\mathcal{A}}(\text{DLog}), \frac{2}{q}\} \).
Verifiable Secret Sharing

Figure 4: The figure above depicts the key stages of noise generation. In the first phase, the randomness generating server shares (depicted above as the server with a bounding box) generate $n_b$ low quality biased bit streams $(b_i^{(1)}, \ldots, b_{n_b}^{(1)})$. It then shares these bits amongst the remaining servers using additive secret sharing. As the protocol is described in a hybrid world, $f_{\text{verify}}$ guarantees that the shares will re-construct to a bit vector in $V$. In the second phase, the servers play Morra to generate one unbiased bit stream for every biased bit stream ($n_b$ unbiased streams in total). These unbiased bits are then $\text{xor}$d with the biased streams to generate an unbiased stream (depicted as $\text{updates}$ in the figure above). However, $\text{xor}$ operation is performed in an arithmetic field instead of a boolean field, and therefore we can leverage commitments to ensure there is no tampering with the shares during aggregation. Finally, each server aggregates its share of unbiased bit streams and broadcasts them to the other servers for one final aggregation.

Proof. The clients are required to commit the inputs given to the servers before the servers draw randomness to verify the validity of the client’s inputs. Additionally, we require that all servers verify if these commitments are correct. By Lemma 5.2, once the shares have been committed, $A$ cannot tamper with them without getting caught by the honest party. They get caught with probability $1 - \text{Adv}_{A}(\text{DLog})$. If the servers do not tamper with the shares, then by the soundness of the sketching protocol, the clients can pass by random chance with probability $\frac{2}{q}$ (as shown in Appendix A).

Unlike Poplar, by Lemma 5.2 and Corollary 5.2.1 we show that we are protected from the attacks described in Fig 3.

6 Covert Tallying Phase

Unless an honest server aborts, we can be confident that we have excluded cheating clients, and the servers have learned no information about an honest client’s input. We still need to ensure that the servers tally their shares correctly and appropriate noise for DP is used. Unlike the semi-honest model, we can no longer delegate noise generation to a single server. Our requirements to satisfy both verifiability and privacy, creates the following two seemingly contradictory conditions: (1) a single honest server should be able to verify if other servers send messages faithfully according to the protocol specification (2) no one can know the randomness used by an honest participant to guarantee DP via the additive mechanism. The second property is needed for us to ensure privacy. Otherwise, an adversary $A$ could subtract the randomness from the output, eliminating any differential privacy guarantees. Therefore we have to design a protocol by which a participant can be certain that the correct magnitude of noise has been applied everyone jointly, without ever knowing the actual value of the noise themselves.

To ensure verifiability, we adapt the distributed binomial mechanism by [44] to make it verifiable and secure under covert security. In their original work, Dwork et al. described distributed protocols to generate Binomial and Laplacian noise with semi-honest participants. Although the Laplacian mechanism would also guarantee differential privacy, it is a non-linear function of its inputs. Thus we cannot easily extend it to ensure correctness. The distributed Binomial mechanism uses linear additive secret shares of Boolean inputs represented in arithmetic fields. Therefore we leverage the additive homomorphic property of commitments to ensure we can guarantee that participants cannot tamper with the protocol specifications. Furthermore, unlike the original algorithm, our public randomness generating game Morra, protects us from active adversaries and gives us verifiability.

Secrecy is ensured by each server independently choosing noise. As long as a single honest server keeps the magnitude of their noise secret, even a computationally unbounded $A$ cannot reconstruct the noise used for differential privacy. Figure 4 summarises protocol $\pi_{\text{agg}}$, which is described in Algorithm 7. It describes how we output verifiable differentially private histograms in the presence of PPT static covert adversaries. To formally prove security, we define the ideal functionality $f_{\text{agg}} : (\mathbb{Z}_q^M)^K \times n \rightarrow \mathbb{Z}_q^M$ as described by (19) as the binomial mechanism in the trusted curator model.
Verifiable Differential Privacy For When The Curious Become Dishonest

Algorithm 7 \( \pi_{agg} \): A protocol for generating verifiable differentially private histograms

**Input:** Each server \( k \in [K] \) receives committed input shares from honest clients \( ([\vec{e}_i^{(k)}], k, t_{ik}) \) for all \( i \in [n] \), number of samples of Bernoulli noise needed \( n_b \).

**Public parameters:** Commitments for all input shares for all \( n \) honest clients, \( g, h \in \mathbb{G}_q \).

**Honest Party Output:** A DP histogram over the honest clients’ inputs.

The protocol proceeds as follows:

1. **Aggregates Inputs:** Each server aggregates its honest client input shares. Let \( B_j = \sum_{i=1}^{n_b} [\vec{e}_{i,j}^{(k)}] \) and \( S_j = \sum_{i=1}^{n_b} t_{ij} \) for \( j \in [K] \).

2. **Input Aggregation:** Once the servers have generated and aggregated noise shares, they combine them with input aggregations. Each server \( j \in [K] \), updates their counters \( B_j = B_j + 1 \) and \( S_j = S_j + 1 \).

3. Each server takes turns to generate \( n_b \) Bernoulli random variables. We refer to server \( k \) as the generating server and the remaining servers as the receiving server. Next, we describe a single run of the protocol for a generating server \( k \in [K] \).

   a. **Low-quality bits:** The generating server samples \( n_b \) values from the set of legal inputs \( V \). More specifically it generates \( \vec{b}_1^{(k)} \ldots \vec{b}_{n_b}^{(k)} \) such that \( \vec{b}_i^{(k)} \leftarrow V \) for all \( i \in [n_b] \).

   b. **Secret Sharing:** Server \( k \) uses committed secret sharing described in Algorithm 5 to share \( \vec{b}_i^{(k)} \) for all \( i \in [n_b] \) among the \( K \) servers. The honest server aborts the protocol if the commitments to their share are not well formed.

   c. **Verification:** Each server receives shares \( \vec{b}_i^{(k)} \), \( s_{ij}^{(k)} \) from the above step. Using calls to \( f_{verify} \), the servers can verify if their received shares add up to a valid value. If for any input \( f_{verify} \) claims \( \vec{b}_i^{(k)} \not\in V \), the protocol is aborted, as one of the servers cheated.

   d. **Generate Unbiased coins:** The servers use \( f_{more} \) to generate public coins \( \vec{c}_1^{(k)} \ldots \vec{c}_{n_b}^{(k)} \) where \( \vec{c}_i^{(k)} = (c_{i1}^{(k)}, \ldots, c_{iM}^{(k)}) \) and \( c_{ij}^{(k)} \sim \text{Bernoulli}(\frac{1}{2}) \) for \( j \in [M] \).

   e. **Linear Updates:** If \( c_{ij}^{(k)} = 0 \) every server leaves \( \text{Index}(\vec{b}_i^{(k)}, j) \) of their share unchanged. If we have \( c_{ij}^{(k)} = 1 \), then just the generating server sets the \( j \)’th index of their corresponding share as follows:

   \[
   \text{Index}(\vec{b}_i^{(k)}, j) \leftarrow 1 - \text{Index}(\vec{b}_i^{(k)}, j) \tag{15} \]

   \[
   \text{Index}(\vec{s}_i^{(k)}, j) \leftarrow 1 - \text{Index}(\vec{s}_i^{(k)}, j) \tag{16} \]

   where \( \text{Index}(\vec{z}, p) \) refers to the \( p \)’th coordinate of \( \vec{z} \). Since the shares have changed, its corresponding commitment is no longer valid and needs to be adjusted. Let \( \gamma = \text{Com} \left( \text{Index}(\vec{b}_i^{(k)}, j), \text{Index}(\vec{s}_i^{(k)}, j) \right) \)

   before the update. If \( c_{ij}^{(k)} = 1 \), then each server updates its view of \( \gamma \) to reflect the change. Note that the commitments are publicly viewable. Thus each server can do this independently. Specifically, the commitments are updated as

   \[
   \gamma \leftarrow g^{1/h^1} \gamma^{-1} \tag{17} \]

   f. **Aggregation:** Each server \( j \in [K] \) updates their counters with the aggregations of their shares i.e., \( B_j = B_j + \sum_{i=1}^{n_b} [\vec{b}_{i,j}^{(k)}] \) and \( S_j = S_j + \sum_{i=1}^{n_b} s_{ij}^{(k)} \). Recall that \( k \) refers to the index of the generating server.

4. **Broadcast:** Finally each server broadcasts to all other servers their counters \( B_j \) and \( S_j \).

5. **Authenticate:** All operations described so far are linear. Thus the servers can use the homomorphic properties of commitments to verify in ciphertext space that the inputs have not been tampered with. Let \( \alpha_{ij}^{(k)} \) and \( \beta_{ij}^{(k)} \) denote the commitment for \( [\vec{b}_{i,j}^{(k)}] \) and \( [\vec{s}_{i,j}^{(k)}] \) respectively. To verify that Server \( j \in [k] \) has not broadcasted incorrect messages, each server checks the following

   \[
   \prod_{i=1}^{n} \prod_{k=1}^{K} \alpha_{ij}^{(k)} \beta_{ij}^{(k)} = g^{B_j} g^{S_j} \tag{18} \]

   If the authentication test fails for a particular \( j \in [K] \), it implies that Server \( j \) has been corrupted. In this phase \( j \) refers to the Server we are verifying and \( k \) refers to the server that generated randomness for this round.

6. **Output:** \( B_1 + \cdots + B_K \) as the answer.
\[ f_{\text{agg}}(\vec{x}_1, \ldots, \vec{x}_K; n) = \sum_{k=1}^{K} (x_k + Y_k) \]  

where \( Y_k = (Y_1, \ldots, Y_M) \) and each \( Y_j \) is sampled i.i.d from a \( \text{Bin}(n_b, \frac{1}{2}) \), where \( n_b \) is defined in Theorem 4.2. Note that if each server \( j \in [K] \) were to use as their input \( x_j^* = \sum_{i=1}^{n} [\vec{\alpha}_i]_j \) then the output is equivalent to adding \( K \) independent copies of appropriately scaled binomial noise to the exact histograms. By Theorem 4.2 we get differentially private histograms.

**Theorem 6.1.** Let \( f_{\text{agg}} \) be the ideal functionality as defined above. Let \( \pi_{\text{agg}}^{(f_{\text{verify}}, f_{\text{more}})} \) represent the protocol described in Algorithm 7 in the hybrid world. Then \( \pi_{\text{agg}}^{(f_{\text{verify}}, f_{\text{more}})} \) securely computes \( f_{\text{agg}} \) in a hybrid world with abort in presence of PPT static covert adversary controlling up to \( K - 1 \) servers with \( \rho \)-deterrent, where \( \rho = 1 - \text{Adv}_A(D\text{Log}) \).

We provide the intuition behind security here. A simulator proof is deferred to Appendix C. Notice all messages sent in the protocol can be described by one of the following:

1. Unconditionally hiding commitments or
2. Aggregations over unconditionally hiding secret shares.

As the protocol is described in the hybrid world, calls to \( f_{\text{verify}} \) ensure that noise shares sent by each server are well-formed, and the verification protocol is secure by definition. Thus all messages can be simulated by uniform samples from \( Z_q \) and \( G_q \); the simulator can easily simulate them. As long as we have a single honest server, the only potential for information leakage is the verification procedure that ensures the shares reconstruct to valid inputs. In the previous section we propose a protocol for securely computing \( f_{\text{verify}} \). Thus we get security by invoking the modular sequential composition theorem described in Theorem 3.1. The protocol is equivalent to each server aggregating additive shares of \( \text{Bin}(n_b, \frac{1}{2}) \) and \( \sum_{i=1}^{n} \vec{v}_i \) and we know that additive secret sharing under our security assumptions is perfectly secure.

The commitments do not add to security but ensure correctness. If the servers used inputs that were not \( \sum_{i=1}^{n} [\vec{\alpha}_i]_k \), \( \sum_{i=1}^{n} \vec{\beta}_i \), and shares of unbiased coins, Lemma 6.2 shows that the honest server would detect this behaviour with probability \( 1 - \text{Adv}_A(D\text{Log}) \). Finally, Theorem 6.3 ensures that unless an honest sever aborts, the noise sampled for the additive mechanism is guaranteed to be drawn from \( \text{Bin}(n_b, \frac{1}{2}) \), where \( n_b \) is as in Theorem 4.2.

**Lemma 6.2.** As long as a single server is semi-honest, if a static PPT covert adversary \( A \) tries to deviate from the protocol specified in Algorithm 7 then the honest server detects such deviation and aborts with probability \( 1 - \text{Adv}_A(D\text{Log}) \).

**Proof.** We will prove the above theorem for \( K = 2 \) as it generalises all values of \( K \geq 2 \). Since each server broadcasts messages to every other server, the output of all honest curators can be combined to describe a single curator indexed by 1. Likewise, we represent all corrupt curators with a single index 2. An adversary could attempt to deviate from the protocol in one of the following ways:

1. \( A \) aborts earlier than prescribed. The honest server always detects this and aborts the protocol.
2. \( A \) controls the generating server \( k \) and samples \( \vec{\beta}^{(k)} \notin V \) for noise generation. Since the protocol is written in the hybrid world, \( f_{\text{verify}} \) would ensure that the honest party would be notified of this deviation.
3. \( A \) could force server 2 to broadcast incorrect messages to the honest server \( \hat{B}_2 \neq B_2 \) and \( \hat{S}_2 \neq S_2 \) instead. This is the only deviation that the honest party does not trivially detect. We next show that \( A \) does so and goes undetected with negligible success probability.

Now assume that \( A \) passes the genuine servers authentication check with non-negligible probability, i.e., that

\[ \prod_{i=1}^{n} \prod_{k=1}^{K} \alpha_{i2}^{(k)} \beta_{i2} = g^{B_2 S_2} = g^{\hat{B}_2 \hat{S}_2} \]  

By Theorem 3.3 this would imply that the \( A \) has a non-negligible advantage in the discrete log game, violating our PPT adversary assumption. \( \square \)
We leave scaling verifiable differential privacy to extremely large dimensions as an open problem. Experiments were

\[
\sum_{k=1}^{K} \vec{Y}_k + \sum_{i=1}^{n} \vec{v}_i
\]

where \( \vec{Y}_k = (Y_1, \ldots, Y_M) \) such that \( Y_j \sim \text{Bin}(n_b, \frac{1}{2}) \) for all \( j \in [M] \).

**Proof.** \( f_{\text{verify}} \) guarantees that \( \vec{b}_i^{(k)} \in V \) and has been shared appropriately among the servers for all generating servers \( k \in [K] \). Let \( b = \text{Index}(\vec{b}_i^{(k)}, j) \) and \( c = \text{Index}(\vec{c}_i^{(k)}, j) \) represent an arbitrary index \( j \in [M] \) and \( i \in [n_b] \). Let \( z = b \oplus c \), then we have that \( z \) is \( \text{Bernoulli}(\frac{1}{2}) \), since by Morra we are guaranteed that each independent coordinate of \( \vec{c}_i^{(k)} \leftarrow R \text{Bernoulli}(\frac{1}{2}) \). Further notice that based on the specification that \( \vec{b}_i^{(k)} \in V \), \( b \) is either 0 or 1. Thus we can write \( z = b \) when \( c = 0 \) and \( z = 1 - b \) when \( c = 1 \) to represent the operation \( z = b \oplus c \). When \( c = 1 \), the generating server subtracts its share \( |b|_k \) from 1, which when combined with the other shares results in \( 1 - \sum_{w=1}^{K} |b|_w = 1 - b \). Thus after the linear update stage, each server possesses shares of \( n_b \text{ Bernoulli}(\frac{1}{2}) \) bits. As the honest server does not abort, we can be certain there has been no tampering with input values. Thus, by additive reconstruction, we get \( \sum_{w=1}^{K} B_k = \sum_{k=1}^{K} \vec{Y}_k + \sum_{i=1}^{n} \vec{v}_i \) which is the desired result.

### 7 Performance

While our protocols are verifiable, we naturally must pay a price in terms of efficiency. In PRIO or Poplar, there are no expensive operations such as exponentiations or group multiplications. In contrast, we have the following extra operations in the three different phases of our protocol: In the secret sharing phase, each client must perform two exponentiations and one multiplication in \( \mathbb{G}_q \) for each share for each dimension. Thus the client performs \( 2KM \) exponentiations and \( KM \) multiplications in \( \mathbb{G}_q \) in total. Each server verifies its share commitments for each client and thus performs a total of \( 2nM \) exponentiations and \( nM \) multiplications. In the input verification phase, each server performs two exponentiations and one multiplication operation for every broadcast message it receives. The server must also adjust the commitments to check if the received messages align. To adjust commitments, the server must perform \( 2M \) exponentiations (one for \( \alpha_i^q \) and one for \( \alpha_i^r \)) and \( 2M \) multiplication operations in \( \mathbb{G}_q \). Finally, for each aggregated message a server receives, it must perform \( (n + n_b)M \) commitment multiplications and \( 2M \) exponentiations. Each server gets \( K \) messages and performs \( O(nKM) \) exponentiation and multiplication operations.

These operations incur a performance cost compared to the non-verifiable construction. Figure 5 compares the running times for secret sharing and input verification. As expected both verifiable and unverifiable privacy have a linear dependence on the number of dimensions \( M \) and the number of servers \( K \). It is clear that even without the optimisation of distributed point functions [26], (unverified) PRIO significantly outperforms our protocol for large values of \( M \). Nevertheless, the total time for our verifiable protocol for dimensions up to a million is less than two minutes in total. We leave scaling verifiable differential privacy to extremely large dimensions as an open problem. Experiments were
run on a M1, 2020, Mac mini with 8GB of RAM, thus it is clear that with larger machines and access to distributed compute, verifiable DP is still practical for small and medium sized $M$. We point out that the loss of performance due to commitments could be avoided using robust secret sharing with Reed-Solomon codes. However, in this case, as pointed out by Or \textit{et al.} \cite{27}, the number of corrupt servers is restricted to being strictly less than $\frac{3}{2}$. Furthermore, robust secret sharing would not protect us from a corrupt server manipulating the verification protocol.

Code and instructions for reproducing these results can be found at https://anonymous.4open.science/r/Verifiable-Differential-Privacy-0407

8 Discussion

The reader might notice that we added $K$ copies of binomial noise (where we expect $K \geq 2$ to be a small constant). In the current scheme, this is unavoidable given we only require one server to be honest. To provide differential privacy, we must guarantee that at least one instance of noise has been added to the output. If we generated noise more efficiently, an adversary controlling the remaining $K - 1$ servers could subtract their share of noise from the output, thereby compromising privacy. Differential privacy achieved via the additive mechanism requires that the noise value remain secret. The authors make the same observation in \cite{29}, and each server independently adds the correct noise calibration. Another observation is that we achieved differential privacy for a specific value of $(\epsilon, \delta)$, corresponding to Bin($n_b, \frac{1}{2}$). It is possible to generate noise in a distributed setting for arbitrary values of $(\epsilon, \delta)$.

Although we show verifiably differentially private computation just for histograms, it is well known, that being able to compute private histograms immediately allows for other important computation. In recent work, Bharadwaj \textit{et al.} show how to compute heavy hitters, succinct histograms and quantiles from histograms \cite{45}.

Finally although we prove security in presence of covert adversaries, we conjecture that with a little more work this model can easily be extended to that of publicly verifiable covert security \cite{46}. Introduced by Orlandi \textit{et al.}, publicly verifiable covert security not only enables the honest party to detect cheating, but they can construct a certificate (without knowledge of the users input) to convince a court of law that the other party did indeed cheat and appropriate actions be taken against them.

References

[1] Varun Raturi, Jinhyun Hong, David Philip McArthur, and Mark Livingston. The impact of privacy protection measures on the utility of crowdsourced cycling data. \textit{Journal of Transport Geography}, 92:103020, 2021.

[2] Robert M Bell and Yehuda Koren. Lessons from the netflix prize challenge. \textit{Acm Sigkdd Explorations Newsletter}, 9(2):75–79, 2007.

[3] Xiaoran Xu, Laming Chen, Songpeng Zu, and Hanning Zhou. Hulu video recommendation: from relevance to reasoning. In \textit{Proceedings of the 12th ACM Conference on Recommender Systems}, pages 482–482, 2018.

[4] Tobias Jeske. Floating car data from smartphones: What google and waze know about you and how hackers can control traffic. \textit{Proc. of the BlackHat Europe}, pages 1–12, 2013.

[5] Danah Boyd and Jayshree Sarathy. Differential perspectives: Epistemic disconnects surrounding the us census bureauâ€™s use of differential privacy. \textit{Harvard Data Science Review (Forthcoming)}, 2022.

[6] Massive data leak exposes 700 million linkedin users’ information. \url{https://fortune.com/2021/06/30/linkedin-data-theft-700-million-users-personal-information-cybersecurity/} Accessed: 2022-07-27.

[7] Uber executive apologises after suggesting the firm dig dirt on hostile journalists. \url{https://www.theguardian.com/technology/2014/nov/18/uber-exec-apologises-emil-michael-journalist-sarah-lacy} Accessed: 2022-07-27.

[8] Simson Garfinkel, John M Abowd, and Christian Martindale. Understanding database reconstruction attacks on public data. \textit{Communications of the ACM}, 62(3):46–53, 2019.

[9] Shiva Prasad Kasiviswanathan, Mark Rudelson, and Adam Smith. The power of linear reconstruction attacks. In \textit{Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms}, pages 1415–1433. SIAM, 2013.

[10] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In \textit{Theory of cryptography conference}, pages 265–284. Springer, 2006.
Verifiable Differential Privacy For When The Curious Become Dishonest

[11] Ashwin Machanavajjhala, Daniel Kifer, John Abowd, Johannes Gehrke, and Lars Vilhuber. Privacy: From theory to practice on the map. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 277–286, 2008.

[12] Samuel Haney, Ashwin Machanavajjhala, John M Abowd, Matthew Graham, Mark Kutzbach, and Lars Vilhuber. Utility cost of formal privacy for releasing national employer-employee statistics. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1339–1354, 2017.

[13] John M Abowd. The us census bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2867–2867, 2018.

[14] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

[15] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 441–459, 2017.

[16] Graham Cormode, Somesh Jha, Tejas Kulkarni, Ninghui Li, Divesh Srivastava, and Tianhao Wang. Privacy at scale: Local differential privacy in practice. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1655–1658, 2018.

[17] Victor Balcer and Albert Cheu. Separating local & shuffled differential privacy via histograms. *arXiv preprint arXiv:1911.06879*, 2019.

[18] Badih Ghazi, Noah Golowich, Ravi Kumar, Rasmus Pagh, and Ameya Velingker. On the power of multiple anonymous messages. *arXiv preprint arXiv:1908.11358*, 2019.

[19] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. The privacy blanket of the shuffle model. In *Annual International Cryptology Conference*, pages 638–667. Springer, 2019.

[20] Wennan Zhu, Peter Kairouz, Brendan McMahan, Haicheng Sun, and Wei Li. Federated heavy hitters discovery with differential privacy. In *International Conference on Artificial Intelligence and Statistics*, pages 3837–3847. PMLR, 2020.

[21] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457, 2001.

[22] Albert Cheu, Adam Smith, and Jonathan Ullman. Manipulation attacks in local differential privacy. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 883–900. IEEE, 2021.

[23] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[24] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. *arXiv:1703.06255 [cs]*, March 2017. arXiv: 1703.06255.

[25] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In *Annual International Cryptology Conference*, pages 67–97. Springer, 2019.

[26] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function Secret Sharing: Improvements and Extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, Vienna Austria, October 2016. ACM.

[27] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 351–371. 2019.

[28] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[29] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight Techniques for Private Heavy Hitters. *arXiv:2012.14884 [cs]*, January 2022. arXiv: 2012.14884.

[30] Alex Davidson, Peter Snyder, EB Quirk, Joseph Genereux, and Benjamin Livshits. Star: Distributed secret sharing for private threshold aggregation reporting. *arXiv preprint arXiv:2109.10074*, 2021.

[31] James Bell, Adria Gascon, Badih Ghazi, Ravi Kumar, Pasin Manurangsi, Mariana Raykova, and Philipp Schoppmann. Distributed, private, sparse histograms in the two-server model. *Cryptology ePrint Archive*, 2022.

[32] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS’07)*, pages 94–103. IEEE, 2007.
Verifiable Differential Privacy For When The Curious Become Dishonest

[33] Zeyu Ding, Daniel Kifer, Thomas Steinke, Yuxin Wang, Yingtao Xiao, Danfeng Zhang, et al. The permute-and-flip mechanism is identical to report-noisy-max with exponential noise. *arXiv preprint arXiv:2105.07260*, 2021.

[34] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[35] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[36] Albert Cheu. Differential privacy in the shuffle model: A survey of separations. *arXiv preprint arXiv:2107.11839*, 2021.

[37] Ælfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by Shuffling: From Local to Central Differential Privacy via Anonymity. *arXiv:1811.12469 [cs, stat]*, July 2020. arXiv: 1811.12469.

[38] Victor Balcer and Albert Cheu. Separating Local & Shuffled Differential Privacy via Histograms. *arXiv:1911.06879 [cs]*, April 2020. arXiv: 1911.06879.

[39] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1253–1269, 2020.

[40] Stanisław Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *Theory of Cryptography Conference*, pages 577–594. Springer, 2009.

[41] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2019.

[42] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[43] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 15(1):23–27, 1983.

[44] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 486–503. Springer, 2006.

[45] Akash Bharadwaj and Graham Cormode. Sample and Threshold Differential Privacy: Histograms and applications, March 2022. arXiv:2112.05693 [cs].

[46] Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 681–698. Springer, 2012.

[47] Badih Ghazi, Noah Golowich, Ravi Kumar, Rasmus Pagh, and Ameya Velingker. On the Power of Multiple Anonymous Messages. *arXiv:1908.11358 [cs, stat]*, May 2020. arXiv: 1908.11358.

[48] Yehuda Lindell. How to simulate it—a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography*, pages 277–346, 2017.
A Proofs for semi-honest sketching

Theorem A.1. Assuming that all \(K \geq 2\) servers are semi-honest, the protocol described in Algorithm 3 satisfies the following properties:

- **Completeness**: If a client’s input \(\vec{v} \in V\), then the servers always admit the input into the protocol.

- **Soundness**: If the vote is illegal i.e., \(\vec{v} \notin V\), then for all corrupt clients (even those that are computationally unbounded), the servers admit the input with cryptographically small probability \(2^{-q}\). This probability can be made arbitrarily small by repeating the protocol in parallel.

- **Zero-Knowledge**: The servers learn nothing more than the information that the client input is valid. More precisely, when \(\vec{v} \in V\), each party can simulate its view of the protocol without ever talking to the other party.

It is easy to see that the above protocol satisfies completeness and soundness:

A.1 Completeness

Let \(j \in [M]\) represent the non zero index of an honest client input. Then \(z^2 = r_j^2\) and \(z^* = r_j^2\). We have \((z^2 - z^*) = 0\), and so a valid input will pass the test.

A.2 Soundness

Let \(\vec{v} \notin V\) represent an illegal input. Observe that verification equation is a degree two polynomial \(p(\vec{r}) = z^2 - z^*\) as described in Equation (22). Also note that we sample \(\vec{r}\) uniformly. Thus, unless we have sampled roots of the polynomial, the left hand side of the equation (22) ensures \(v_i \in \{0, 1\}\) and the right hand side ensures the uniqueness of 1 non-zero \(v_i\).

\[
z^2 - z^* = \left( \sum_{i \in [M]} r_i v_i \right)^2 - \sum_{i \in [M]} r_i^2 v_i = \sum_{i \in [M]} r_i^2 v_i (v_i - 1) + 2 \sum_{i,j : i \neq j} r_i r_j v_i v_j \tag{22}
\]

The probability of accidentally sampling roots is \(2^{-q}\) as the verification polynomial is quadratic and the sampling procedure is uniform. Thus we get soundness probability \(2^{-q}\), which can be made negligible by repeating this process multiple times.

A.3 Zero Knowledge

To show that the semi-honest servers do not learn any information about the client’s input, we must be able to construct a simulator in the ideal world whose view is indistinguishable from the real-world adversary. We define the ideal functionality \(f_{\text{verify}} : \mathbb{Z}_q^K \rightarrow \{0, 1\}\) as the following

\[
f_{\text{verify}}(x_1, \ldots, x_k) = \begin{cases} 1 & \text{if } (\sum_{i=1}^{K} x_k) \in V \\ 0 & \text{otherwise} \end{cases}
\]

During the broadcast phase, each server receives a message from every other server. Given we always assume at least one honest server, we can generalise to the \(K \geq 2\) setting by proving security for \(K = 2\). Let \(P_0\) denote the single honest server. We can aggregate messages from all other servers and pretend they come from a single source \(P_1\). Each party’s input on their tape is denoted by \([\vec{v}]_0\) and \([\vec{v}]_1\), respectively. They send their inputs to a trusted third party.

As \(f_{\text{verify}}\) is a deterministic function, and we have already established the correctness property of our protocol, it is sufficient to simulate the view of the real-world adversary instead of the joint distribution of view and output of the protocol. Let \(\text{Sim}\) denote the simulator interacting with the corrupted party \(P_1\) in the ideal world. As the adversaries are semi-honest, \(\text{Sim}\) has access to \([\vec{v}]_1\) and does not need to extract effective inputs. The simulator proceeds as follows:
1. $S_0$ samples $\vec{r} = (r_1, \ldots, r_M) \leftarrow \mathbb{Z}_q^M$. It then computes $\vec{r}^* = (r_1^2, \ldots, r_M^2)$.

2. Note that by the specifications of the voting phase of the protocol, $[\vec{v}]_1 \leftarrow \mathbb{Z}_q^M$ and is independent of $[\vec{v}]_0$, thus the simulator simply samples a random value $\vec{x} \sim \mathbb{R}^V$ and sets $[\vec{x}]_0 = \vec{x} - [\vec{v}]_1$. It then pretends to be the honest server and interacts with the adversary $\mathcal{A}$ controlling $P_1$. Specifically, it computes $\vec{z}_0 = \langle \vec{r}, [\vec{x}]_0 \rangle$ and $\vec{z}_0^* = \langle \vec{r}^*, [\vec{x}]_0 \rangle$ sends them to $\mathcal{A}$ simulating the honest server and outputs whatever $\mathcal{A}$ outputs.

3. If $\mathcal{A}$ does not respond, then $\text{Sim}$ sends $\text{ABORT}_1$ to the ideal functionality, outputs whatever $\mathcal{A}$ outputs and aborts simulating the honest party.

B The Binomial mechanism

In this Appendix, we spell out the details of the differential privacy properties of Binomial noise addition (the Binomial mechanism). The results here were originally shown by Ghazi et al. [47], and we include them here for completeness (we do not claim any novelty in this section).

**Definition 3.** [47] A function $q : \mathcal{X}^n \to \mathbb{Z}^M$ is said to be $k$-incremental if for all neighbouring datasets $X \sim X'$, $||f(X) - f(X')||_\infty \leq k$.

It is easy to see that counting queries for histogram estimation are 1-incremental. The following definition describes valid noise distributions to ensure differential privacy.

**Definition 4.** [47] A distribution $\mathcal{D}$ over $\mathbb{Z}$ is $(\epsilon, \delta, k)$-smooth if for all $k' \in [-k, k]$ we have

$$\Pr_{Y \sim \mathcal{D}} \left[ \Pr_{Y' \sim \mathcal{D}}[Y' = Y] \geq e^{\epsilon k'} \right] \leq \delta$$

(24)

The result for the Binomial mechanism follows by showing that adding noise drawn from a smooth distribution ensures differential privacy and then showing that the Binomial distribution meets the smoothness definition.

**Lemma B.1** (Lemma 4.11 in Appendix C of [47]). Suppose $q : \mathcal{X}^n \to \mathbb{Z}^M$ is $k$-incremental i.e., for all neighboring datasets $X \sim X'$ we have $||q(X) - q(X')||_\infty \leq k$ and $\Delta(q) = ||q(X) - q(X')||_1 = \Delta$. Let $\mathcal{D}$ be a $(\epsilon, \delta, k)$-smooth distribution. Then the mechanism $\mathcal{M}$

$$\mathcal{M}(Y_1, \ldots, Y_M)(X, q) = q(X) + (Y_1, \ldots, Y_M)$$

(25)

is $(\epsilon \Delta, \delta \Delta)$ differentially private, where $(Y_1, \ldots, Y_M) \sim \mathcal{D}$.

**Proof.** Let $X = (x_1, \ldots, x_n)$ where $x_i \in \mathcal{X}$ and $X' = (x_1', \ldots, x'_n)$. Let $\vec{y} = (y_1, \ldots, y_M)$ and $\vec{y}' = (y_1', \ldots, y_M)$ be i.i.d draws from $\mathcal{D}$. Assume that Equation (26) holds:

$$\Pr_{\vec{y} \sim \mathcal{D}}[g(\vec{y}) \geq e^{\epsilon'}] \leq \delta'$$

(26)

where $g(\vec{y}) = \Pr_{\vec{y} \sim \mathcal{D}}[M_{\vec{y}}(X, q) = q(X) + \vec{y}] / \Pr_{\vec{y} \sim \mathcal{D}}[M_{\vec{y}}(X, q) = q(X)]$.

Let $S \subseteq \mathbb{Z}^M$ be an arbitrary subset in the range of $\mathcal{M}$. Let $T = \{M_{\vec{y}}(X, q) | g(\vec{y}) < e^{\epsilon'} \}$ represent a set of outputs of $\mathcal{M}$ over draws of $\vec{y}$ such that $g(\vec{y}) < e^{\epsilon'}$. Then from equation (26) we can show that $\mathcal{M}$ is $(\epsilon', \delta')$ differentially private

$$\Pr_{\vec{y} \sim \mathcal{D}}[M_{\vec{y}}(X, q) \in S] \leq \delta' + \sum_{w \in S \cap T} \Pr_{\vec{y} \sim \mathcal{D}}[M_{\vec{y}}(X, q) = w]$$

(27)

$$\leq \delta' + \sum_{w \in S \cap T} e^{\epsilon'} \Pr_{\vec{y} \sim \mathcal{D}}[M_{\vec{y}}(X', q) = w]$$

(28)

$$\leq \delta' + \sum_{w \in S} e^{\epsilon'} \Pr_{\vec{y} \sim \mathcal{D}}[M_{\vec{y}}(X', q) = w]$$

(29)

$$= \delta' + e^{\epsilon'} \Pr_{\vec{y} \sim \mathcal{D}}[M_{\vec{y}}(X', q) \in S]$$

(30)
Equation (27) is from the law of total probability, equation (29) comes from equation (26) assumption and equation (30) is true as \( T \cap S \subseteq S \). Therefore all that remains is to show that equation (26) is true if \( \mathcal{D} \) is as defined and \( \epsilon' = \epsilon \Delta, \delta' = \delta \Delta \) to complete the proof.

Define \( k_j = q(X)_j - q(X')_j \). As each coordinate of \( q(X) \) is independently perturbed and \( q \) is a deterministic function, equation (26) is equivalent to equation (32).

Thus, in order to prove equation (26) is true, it suffices to show that equation (32) holds. We know that \( \mathcal{D} \) is a smooth distribution i.e for each \( j \in [M] \)

\[
\Pr_{\tilde{y} \sim \mathcal{D}} \left[ \prod_{j=1}^{M} \frac{\Pr_{(Y_j \sim \mathcal{D})} (Y_j = y_j)}{\Pr_{(Y_j \sim \mathcal{D})} (Y_j = y_j + k_j)} \geq e^{\epsilon k_j} \right] \leq \delta' \tag{32}
\]

We can apply the union bound to get the probability of the joint distribution over all indices.

\[
\Pr_{\tilde{y} \sim \mathcal{D}} \left[ \prod_{j=1}^{M} \frac{\Pr_{(Y_j \sim \mathcal{D})} (Y_j = y_j)}{\Pr_{(Y_j \sim \mathcal{D})} (Y_j = y_j + k_j)} \geq e^{\sum_{j=1}^{M} |k_j| \epsilon} \right] \leq \delta \sum_{j=1}^{M} \mathbb{I}(k_j \neq 0) \tag{34}
\]

Given the sensitivity of \( q \) is \( \Delta \), at most \( \Delta \) indices for \( k_j \) can be non-zero and \( \sum_{j=1}^{M} |k_j| \leq \Delta \). Finally we get the result we seek

\[
\Pr_{\tilde{y} \sim \mathcal{D}} \left[ \prod_{j=1}^{M} \frac{\Pr_{(Y_j \sim \mathcal{D})} (Y_j = y_j)}{\Pr_{(Y_j \sim \mathcal{D})} (Y_j = y_j + k_j)} \geq e^{\Delta |k_j| \epsilon} \right] \leq \delta \Delta \tag{36}
\]

**Lemma B.2** (Based on Lemma 4.12 of Appendix C in [47]). Let \( n \in \mathbb{N}, p \in [0, 1/2], \alpha \in [0, 1) \) and \( k \leq \frac{n p}{2} \). Then the binomial distribution \( \text{Bin}(n, p) \) is a \((\epsilon, \delta, k)\)-smooth distribution.

**Proof.** Let \( Y \sim \text{Bin}(n, p) \), then \( \Pr[Y = y] = \binom{n}{y} p^y (1 - p)^{n-y} \). For any \(-k \leq k' \leq k\), define an interval \( \epsilon := [(1-\alpha)np + k', (1+\alpha)np - k'] \). This an interval of size \( k \) around the mean of the distribution. Note that as long as \( k \leq \frac{n p}{2} \alpha \), then the interval \( \epsilon' := [(1/2)p - (1-\alpha/2)np, (1+\alpha/2)np - (1/2)p] \) is contained inside of \( \epsilon \). Thus if \( Y \sim \text{Bin}(np) \) is not in \( \epsilon \), it is also not inside \( \epsilon' \). We know how to bound the probability that \( y \notin \epsilon' \) by using the multiplicative Chernoff bound. Invoking it, we get

\[
\Pr_{y \sim \text{Bin}(n, p)} [y \notin \epsilon] \leq \Pr_{y \sim \text{Bin}(n, p)} [y \notin \epsilon'] \\
\leq e^{-\frac{\alpha^2 np}{8}} + e^{-\frac{(1+2\alpha) np}{8+4\alpha}} = \delta
\]

Now for all \( y \in \epsilon \), we have for \( 0 \leq k' \leq k \)

\[
\Pr[Y = y] / \Pr[Y = y + k'] = \left( \frac{1-p}{p} \right)^{k'} \prod_{i=1}^{k'} \frac{y + i}{n - y - i + 1} \tag{37}
\]

\[
\leq \left( \frac{1-p}{p} \right)^{k'} \left( \frac{y + k'}{n - y - k'} \right)^{k'} \tag{38}
\]

\[
\leq \left( \frac{1-p}{p} \right)^{k'} \left( \frac{(1+\alpha)np}{n - (1+\alpha)np} \right)^{k'} \tag{39}
\]

\[
= (1+\alpha)^{k'} \left( \frac{1-p}{1-\alpha p} \right)^{k'} \tag{40}
\]
Verifiable Differential Privacy For When The Curious Become Dishonest

(39) comes from our assumption \( y \in \epsilon \) and so when \( y = (1 + \alpha)np - k' \) the ratio above is maximal. Similarly for \(-k \leq k' \leq 0\) we have

\[
\frac{\Pr [Y = y]}{\Pr [Y = y + |k'|]} = \left( \frac{p}{1 - p} \right)^{|k'|} \prod_{i=1}^{|k'|} \frac{n - y + i}{y - i + 1} \text{ (41)}
\]

\[
\leq \left( \frac{p}{1 - p} \right)^{|k'|} \left( \frac{n - y + |k'|}{y - |k'|} \right)^{|k'|} \text{ (42)}
\]

\[
\leq \left( \frac{1 + np - p}{(1 - \alpha)(1 - p)} \right)^{|k'|} \leq \left( \frac{1 + \alpha}{1 - \alpha} \right)^{|k'|} \text{ (43)}
\]

\[
= e^{\epsilon |k'|} \text{ (44)}
\]

(45) comes from plugging in the smallest value for \( y \) and (44) comes from the fact that \( p \leq 1/2 \). It is easy to see that when \( p \leq \frac{1}{2} \), that we have

\[
\left( \frac{1 - p}{1 - p - p\alpha} \right)^{k'} \leq \left( \frac{1}{1 - \alpha} \right)^{|k'|} \text{ (46)}
\]

Therefore we can upper bound equation (40) by setting

\[
(1 + \alpha)^{k'} \left( \frac{1 - p}{1 - p - p\alpha} \right)^{k'} \leq e^{\epsilon |k'|} \text{ (47)}
\]

Finally, we can prove smoothness using Bayes’ rule. Let \( g(y) = \frac{\Pr_{Y \sim \text{Bin}(n, p)}[Y = y]}{\Pr_{Y \sim \text{Bin}(n, p)}[Y = y + k']} \) and \( D = \text{Bin}(n, p) \). Then:

\[
\Pr_{y \sim D} [g(y) \geq e^{\epsilon |k'|}] \leq \Pr_{y \sim D} [g(y) \geq e^{\epsilon |k'|} | y \notin \epsilon] + \delta \leq e^{-\frac{\alpha^2 np}{\alpha + 2\alpha}} + e^{-\frac{\alpha^2 np}{\alpha + 2\alpha}} \leq \delta \text{ (48)}
\]

(48) comes from the fact that when \( y \in \epsilon \) we have \( \Pr_{Y \sim \text{Bin}(n, p)}[Y = y] \leq e^{\epsilon |k'|} \) by how we defined \( \epsilon \) in equation (44). So we have

\[
\Pr_{y \sim \text{Bin}(n, p)} \left[ \frac{\Pr_{Y \sim \text{Bin}(n, p)}[Y = y]}{\Pr_{Y \sim \text{Bin}(n, p)}[Y = y + k']} \geq e^{\epsilon |k'| | y \notin \epsilon} \right] = 0
\]

\[\square\]

C Covert Security Definitions

The adversary \( A \) controls up to \( K - 1 \) servers in all our protocols. Thus all protocols can be represented by a two-party protocol where \( A \) controls one party, referred to as \( P_1 \). Without loss of generality, we assume that the other party, referred to as \( P_0 \), is semi-honest. Although we prove security for two-party protocols, given the assumptions above, the proofs generalise for \( K \geq 2 \) for all the protocols in this paper. We use the explicit cheat formulation of covert security from [28] as the definition of covert security. Under this formulation, the ideal model proceeds as follows:

1. **Inputs:** Each party obtains an input denoted by \( x_0 \) and \( x_1 \). We assume that inputs are of the same length. The real-world adversary \( A \) receives an auxiliary input \( z \).

2. **Send Inputs to Trusted Party:** \( P_0 \) always sends \( x_0 \) to the trusted party but \( P_1 \) is controlled by a covert adversary \( A \), it could send any value \( x \) that is a PPT function of its true input \( x_1 \) and auxiliary input \( z \). Let \( w \) denote the vector of inputs sent to the trusted party from \( P_1 \) and \( P_2 \).
3. **Abort Options:** If $P_1$ had sent $\text{ABORT}$ to the trusted party. The trusted party responds to all parties with $\text{ABORT}$ and halts. If $P_1$ had sent $\text{CORRUPT}$ to the trusted party. The trusted party responds to all parties with $\text{CORRUPT}$ and halts.

4. **Attempted Cheat Option:** If $P_1$ had sent $\text{CHEAT}$ to the trusted party, then the trusted party reveals $x_0$ to $P_1$. However, in addition to this, with probability $\rho$, the trusted party sends $\text{CORRUPT}$ to $P_0$. With probability $1 - \rho$ it sends $\text{UNDETECTED}$ to $P_1$, following which $P_1$ sends the trusted party $y_0$ to send to the honest party instead of the correct output $f_0(\bar{w})$. The ideal execution ends at this point. If there was no $\text{ABORT}$, $\text{CHEAT}$, or $\text{CORRUPT}$, the ideal execution continues.

5. **Trusted Party Replies to Adversaries:** The trusted party sends sends $f_1(\bar{w})$ to the adversary.

6. **Trusted Party Replies to Honest Party:** After receiving messages from the trusted party, $P_1$ sends the trusted party either $\text{ABORT}$ or $\text{CONTINUE}$. If the trusted party received $\text{CONTINUE}$ then it sends $f_0(\bar{w})$ to the honest party. Otherwise, it sends $\text{ABORT}$ to the honest party.

7. **Output:** The honest party always outputs whatever is received from the trusted party. The adversary $\mathcal{A}$ controlling the other party is allowed to output any PPT function of the initial input $x_1$, the auxiliary input $z$ and messages received from the trusted party.

**Theorem C.1.** Let $\mathcal{B}$ represent an honest client’s input and $f_{\text{verify}}$ denote the deterministic function that checks if $v \in V$. Let $\pi_{\text{sketch}}$ refer to the protocol described in Algorithm 7. Then $\pi_{\text{sketch}}$ securely computes $f$ with abort in the presence of static covert adversary with $\rho$-deterrent, where $\rho = 1$.

**Proof.** We will prove the above theorem for $K = 2$ as it generalises all values of $K \geq 2$. The output of all honest servers can be combined to describe a single curator indexed by 0. Likewise, we represent all corrupt curators with a single index 1. For deterministic ideal functionalities such as $f_{\text{verify}}$, it is enough to simulate the adversary’s view as long as the protocol is correct (see Appendix A for proof of correctness); thus, it is sufficient to simulate the adversary’s view $\mathcal{A}$ instead of the joint distribution of the view and the output. Let $\left([\bar{v}]_0, \bar{t}_0\right)$ denote the honest server’s input and let $\left([\bar{v}]_1, \bar{t}_1\right)$ denote the input of the server controlled by a static PPT covert adversary $\mathcal{A}$. We follow the convention introduced by Lindell’s simulation tutorial [48, remark 6.2] and assume that $\mathcal{A}$ is deterministic and the simulator picks the input, random and auxiliary tape for the adversary. We construct the following simulator $\text{Sim}$ that internally interacts with $\mathcal{A}$ and externally with $f_{\text{verify}}$:

1. Sim receives $\left([\bar{v}]_1, \bar{t}_1\right)$ on its input tape. This is the input Server 1 would have received, but as it is corrupted by a covert adversary $\mathcal{A}$, it may use a different value in the real-world protocol. Sim must therefore be able to extract the effective inputs used by $\mathcal{A}$ to simulate $\mathcal{A}$’s view. Along with the inputs, it also receives on its auxiliary tape the public commitments of all shares.

2. Sim selects a random $\bar{r} \leftarrow R \mathbb{Z}_q^M$ from the set of legal inputs and sets $\left([\bar{r}]_0 = \bar{r} - [\bar{v}]_1\right)$. It also samples $\bar{t}_0 \leftarrow R \mathbb{Z}_q^{M}$ and computes $\text{Com}(\left([\bar{x}]_0, \bar{t}_0\right))$.

3. Sim pretends to be the honest server and follows $\pi_{\text{sketch}}$ with input $\left([\bar{x}]_0, \bar{t}_0\right)$. Specifically, it samples $\bar{r} = (r_1, \ldots, r_M) \leftarrow R \mathbb{Z}_q^M$ and sends to $\mathcal{A}$, the values $u_0$, $z_0$ and $z^*_0$, where $z_0 = \langle \bar{r}, [\bar{x}]_0 \rangle$, $u_0 = \langle \bar{r}, \bar{t}_0 \rangle$, $z^*_0 = \langle \bar{r}^*, [\bar{x}]_0 \rangle$. If $\mathcal{A}$ does not respond, Sim sends the ideal functionality $\text{ABORT}_1$, which notifies all parties of such behaviour. Otherwise Sim receives $z_1$ and $z^*_1$ from $\mathcal{A}$. Let $z = z_0 + z_1$ and $z^* = z^*_0 + z^*_1$. If $\mathcal{A}$ passes both the sketching test and authentication test, then Sim accepts that $\mathcal{A}$ has not cheated and outputs whatever $\mathcal{A}$ outputs. Note that it does not matter that $\text{Sim}$ did not have access to the honest party’s input $\left([\bar{v}]_0, \bar{t}_0\right)$ since $[\bar{x}]_0$ and $[\bar{v}]_0$ are perfectly indistinguishable—$\mathcal{A}$ has no way of distinguishing between them. Therefore it employs the same strategy against both. $\mathcal{A}$ deviates from the protocol and still passes both tests with probability at most $\text{Adv}_{\mathcal{A}}(\text{DLog})$ as shown in Lemma 5.2 which by assumption is negligible.

4. If $\mathcal{A}$ fails any of the tests, then $\text{Sim}$ sends $\text{CHEAT}_1$ to the ideal functionality, which then reveals $[\bar{v}]_0$ to Sim and sends the honest party $\text{CORRUPT}_1$ with probability $\rho = 1$. Sim now possesses $[\bar{v}]_0$ and so can simulate $\mathcal{A}$’s output by carrying out the protocol as specified. So it rewinds $\mathcal{A}$ as if it had never sent anything and then pretends to be the honest server and continues the protocol. In active security, where the ideal world does not have a specific cheat token, the simulator would not be able to simulate the real-world adversary’s output.

23
Theorem C.2. Let $f_{\text{agg}}$ be the ideal functionality as defined above. Let $\pi_{\text{agg}}(f_{\text{verify}}, f_{\text{morra}})$ represent the protocol described in Algorithm 7 in the hybrid world. Then $\pi_{\text{agg}}(f_{\text{verify}}, f_{\text{morra}})$ securely computes $f_{\text{noise}}$ in a hybrid world with abort in presence of PPT static covert adversary controlling upto $K-1$ servers with $\rho$-deterrent, where $\rho = 1 - \text{Adv}_A(D\text{Log})$.

Proof. We construct the following simulator.

1. Sim receives on its input tape the $[v_i^1]_1$ and $t_i^1$ for all $i \in [n]$ where $n$ is the number of honest clients. It receives on its auxiliary tape the public parameters. Sim has no idea what the actual values for $[v_i^0]_0$ and $t_i^0$ are. So instead it samples $v_i^1 \vec{\in} V$ and sets $[v_i^0]_0 = v_i^1 - [v_i^1]_1$ and $t_i^0 \vec{\in} \mathbb{Z}_q$ for all $i \in [n]$. Additionally, it creates commitments for each share, just like a client would. The simulator then hands over $\{[v_i^1]_1, t_i^1\}_{i=1}^n$ to $A$ and makes the commitments public. It uses $\{[v_i^0]_0, t_i^0\}_{i=1}^n$ to pretend to be the honest party and completes the protocol as prescribed by internally interacting with $A$.

2. As we operate in the hybrid world, we can assume that the generating server sampled $b_i^k$ correctly for all $i \in n_b$. Note that $A$ interacts with $f_{\text{verify}}$ via the Simulator in the ideal world. Thus any tampering would be caught by the Simulator, who would then send CORRUPT to the trusted party.

3. The Simulator samples unbiased coins $(c_1^{(k)}, \ldots, c_{n_b}^{(k)})$ by participating in $M n_b$ rounds of Morra with $A$. Again as we have access to $f_{\text{morra}}$, we can safely assume this is done securely.

4. At this stage, the Simulator has all the information to update its commitments based on the public coins. It broadcasts the aggregation of inputs and noise to the adversary. If $A$ does not respond, it sends ABORT to the trusted party. If $A$ sends messages that fail the authentication test, it sends CORRUPT to the trusted party. Otherwise, it continues the protocol as prescribed, pretending to be the honest server. Note that $A$ passes the authentication test but still tampers with inputs with probability $1 - \text{Adv}_A(D\text{Log})$. Thus with probability $\text{Adv}_A(D\text{Log})$, its cheating goes undetected.

Note that Sim never used the CHEAT token. This is because the only potential leak of privacy occurs during verification. The rest of the protocol is secure against active adversaries.

\[\square\]

D Discrete Log Assumption

The discrete log game consists of interactions between a challenger and an adversary PPT $A$. The game proceeds as follows:

1. The challenger samples $x \vec{\in} \mathbb{Z}_q$ and computes $h = g^x$
2. The challenger sends $h, g$ to $A$ over a secure authenticated channel
3. The adversary tries to guess $x$ by outputting $\hat{x}$

We define the advantage of the adversary as the probability of guessing $x$ correctly i.e.

\[\text{Adv}(A, G) := \Pr[\hat{x} = x]\]

The discrete log assumption states that for all probabilistic polynomial time adversaries $A$, $\text{Adv}(A, G) = \mathcal{O}(q)$ where $\mu$ is negligible in $q$. 

24