Zero-knowledge Differential Privacy: Theory and Implementation *

Emily Kang † Sol Kim ‡ Jaray Liu § Max Wagner ¶ Weiyuan Gong

Abstract

Differential Privacy (DP) has become a widely adopted standard for the release of aggregate statistics over sensitive data. However, DP's reliance on randomness introduces a subtle risk: a malicious data curator could intentionally manipulate results while blaming discrepancies on noise. To address this issue, Biswas and Cormode [1] proposed a theoretical framework of verifiable DP, where the data curator must produce a zero-knowledge proof certifying that the reported output is both differentially private and correctly computed. This ensures verifiability without revealing the private randomness or compromising privacy. Nevertheless, the instantiation of their verifiable DP counting queries relies heavily on homomorphic commitments and *unbiased* public coin flipping to achieve DP using *unbiased binomial mechanism*. In this project, we propose a similar protocol to achieve a *biased binomial mechanism*. To this means, we construct a *verifiable biased public coin flipping protocol*. We also further propose a zero-knowledge randomized response protocol satisfying verifiable DP, which does not rely on the homomorphic property of commitments. Finally, to demonstrate zero-knowledge to a broader audience, we implement an interactive interface in which a malicious prover can attempt to cheat throughout the protocol for the unbiased binomial mechanism.

1 Introduction

The privacy and security research community has developed a range of privacy-enhancing technologies to safeguard individuals whose data is included in published statistics. Among these, Differential Privacy (DP) and its variants are particularly significant. DP ensures that statistical computations are randomized to limit the impact of any single individual's data on the output, thereby protecting privacy. This is commonly achieved by injecting carefully controlled random noise into the results, which inevitably reduces accuracy. The standard formulation of DP operates under the trusted curator model, where a single trusted entity collects the raw data and is responsible for applying the randomized algorithm.

In a recent work, Biswas and Cormode [1] introduced a framework for Interactive Proofs for Differentially Private Counting, designed to address vulnerabilities in conventional differential privacy (DP) mechanisms. The problem statement is that standard DP techniques add calibrated random noise to data aggregates to protect individual privacy, but this noise can be exploited by a malicious aggregator. By masking intentional deviations in computed statistics behind the randomness inherent in DP, an adversary might skew aggregate outputs without being detected. Therefore, the paper reinterprets DP in an adversarial setting by introducing interactive proofs into the DP framework. This means that besides

*Course project of 2025 Spring Harvard CS 2080.

†Email: emily_kang@college.harvard.edu.
‡Email: solkim@college.harvard.edu.
§Email: jarayliu@college.harvard.edu.
¶Email: maxwagner@college.harvard.edu.
Email: wgong@g.harvard.edu.

releasing a noisy aggregate, the server or aggregator must also provide a zero-knowledge (ZK) proof that the output is both generated according to the prescribed DP algorithm and that the randomness (noise) was sampled faithfully. In order to combat this possibility, Biswas and Cormode introduced a method of privately noising data using some input from public noise, in a way in which it is provable that the noise was added. Thus, an outside observer can be confident that the DP protocol was correctly applied and thus put more faith in outcomes drawn from the resulting data.

They further concrete instantiations of verifiable DP by computing DP counting queries (histograms) in the trusted curator and client-server multiparty settings. From a high-level perspective, given n data samples $X = (x_1, ..., x_n) \in \mathbb{Z}_q$ from n clients, the goal of the servers (also denoted as provers) is to differentially privately output the counting query $Q(X) = \sum_{i=1}^{n} x_i$. To achieve DP, the servers sample n_b *unbiased* coins and output $y = Q(X) + \Delta_k$ for $\Delta_k \sim \text{Binomial}(n_b, 1/2)$. To verify that the reported output is both differentially private and correctly computed, they design a public unbiased coin flipping protocol among clients, provers, and a verifier based on cryptographic tools such as Pedersen Commitment schemes [2]. We recap the concept, the formal definitions, and the design of the algorithm in Section 4.

While Biswas and Cormode provide an interactive proof for the unbiased Binomial mechanism, there is great potential for work to be done in terms of (i) extending verifiability to other canonical mechanisms (ii) reducing the cryptographic assumptions of these protocols, and (iii) producing open source implementations that increase accessibility and transparency around DP. The sections that follow tackle each of these gaps, first by giving an interactive proof for the biased Binomial mechanism and then by presenting a zero knowledge randomized-response protocol that eliminates the need for homomorphic commitments altogether. We thus ask,

How can we expand a fair verifiable Binomial mechanism towards broader applications of ZKDP?

1.1 Our results

1.1.1 Verifiable DP via biased binomial mechanism

We take a step forward along this question by proposing a verifiable protocol computing DP counting queries (histograms) in the single trusted curator model using *biased* coin flipping. In particular, the servers sample n_b *unbiased* coins and output $y = Q(X) + \Delta_k$ for $\Delta_k \sim \text{Binomial}(n_b, p)$ for any p, $X = (x_1, ..., x_n)$, and $Q(X) = \sum_{i=1}^{n} x_i$ to satisfy DP and the verifiers certify the provers honestly carry out the protocol using commitments.

Theorem 1.1 (Informal, see Theorem 5.1)**.** *Let $X = (x_1, ..., x_n)$ be the client input. There is a verifiable DP protocol in the output $y = Q(X) + \Delta_k$ in the single trusted curator model, where $\Delta_k \sim \text{Binomial}(n_b, p)$ for any p and $Q(X) = \sum_{i=1}^{n} x_i$.*

We note that this is not a direct extension by replacing the public version of sampling *unbiased* coins with sampling *biased* coins. This is because the corrupted provers can make use of the biased target distribution and control the final distribution by dishonestly generating their own private randomness. To address this issue, we design a public *biased* coin flipping protocol that can be verified using Pedersen Commitments that guarantee exclusion of such dishonest behavior. We leave the detailed protocol and the proof of verifiable DP in Section 5.

1.1.2 Zero knowledge randomized response

We attempt to give an implementation of a zero-knowledge proof for general p. This brings us towards our goal of relaxing the minimum assumptions because we do not utilize the homomorphic commitment scheme property to verify randomized response.

1.1.3 Accessible implementations of verifiable DP

Along with this theoretical development of a zero-knowledge mechanism for verifying randomized response, we implemented the zero-knowledge proof of the binomial mechanism for DP as a rust backend server, accessible through a frontend at zkdp.org. This website is intended at educating about the mechanism behind Zero-Knowledge proofs, specifically pertaining to differential privacy, and fully implements each step of the proof, as well as opportunities for the user (playing as the prover), to attempt to "cheat" by unfairly modifying values. After this, the site will show the user where they were caught in the process. In the code, based on Ari Biswas' work in Rust, and modified to work as a single callable structure, with a GET/POST interface for web requests, commitments are modeled based on the Ristretto method of constructing prime-order elliptic curve groups to generate one-way commitments. On the site, the server functions as the verifier, and checks the user's inputs in order to ensure that no cheating has happened before allowing a proved release.

2 Discussion and future work

Our work primarily improves upon the work of Biswas and Corder [1]. While Biswas and Corder show that the unbiased Binomial mechanism is verifiably DP, we extend further and show the biased Binomial mechanism ($p \neq \frac{1}{2}$) is verifiably DP under the *same* minimum assumptions used in [1]. Specifically, under the assumption of one-way homomorphic functions, we provide a verifiable method of generating n draws from a Bern(p) distribution for some $p = \frac{k}{m}$.

Furthermore, we show that the Randomized Response protocol is verifiably DP *without* the assumption of homomorphic commitment schemes, reducing the minimum assumptions of Biswas and Corder.

Nonetheless, our work is not without shortcomings. First, we still rely on a trusted setup for the commitment scheme and on the hardness of discrete log for Pedersen commitments; a break in either would violate zero knowledge and invalidate the proof. Second, while our biased mechanism works if either the prover or verifier is malicious, our work fails if both parties are corrupt and collude with each other. As an example, if the prover tells the verifier which k bits are 1, then the verifier could control exactly how much noise is added, thus breaking privacy. Notably, the unbiased mechanism of Biswas and Cormode [1] also suffers from this limitation. Finally, our interactive proofs incur substantial communication overhead, most notably in the biased Binomial routine. Committing the m bit expansion of p for each of the n_b draws alone inflates the transcript to $O(mn_b)$ commitments and accompanying Σ-OR proofs.

These limitations yield several immediate directions forward. First, one could attempt to adapt our randomized response protocol to the Binomial case, removing the need for one way homomorphic functions entirely. Second, the issue of joint corruption can be mitigated by distributing the role of the prover among several independent parties, so that no single participant can control or bias the added noise. While a malicious prover could still leak the raw data and break privacy, such a modification to our zero knowledge protocol would at least guarantee that any released statistic was produced faithfully even if both parties collude. Third, the biased p Algorithm could be rewritten using one-way homomorphic vector commitment schemes which are much more efficient, thus increasing the efficiency of the iteractive proof. Beyond these fixes, there still remains the vast majority of DP mechanisms (Gaussian, Laplace, etc.) which still lack zero knowledge protocols, offering promising milestones for future ZKDP work.

3 Preliminaries

Notations. Throughout this paper, following the similar notation as Ref. [1], we will write $x \xleftarrow{R} U$ to denote that x was uniformly sampled from a set U. We denote data vectors as $\vec{x} \in \mathbb{Z}_q^n$, where M is the size of the vector and $\mathbb{Z}_q$ is a prime order finite field of integers of size q.

3.1 Commitments

Commitments are widely employed in previous and our schemes to ensure that participants cannot behave dishonestly by changing their response during the protocol. We formally define it as follows:

Definition 3.1 (Commitments)**.** *Given security parameter $\kappa \in \mathbb{N}$, a non-interactive commitment contains a pair of probabilistic polynomial-time algorithms $(Setup, Com)$. Here, the setup algorithm $pp \leftarrow Setup(1^\kappa)$ generates public parameters pp. The commitment algorithm Com_{pp} defines a function $M_{pp} \times R_{pp} \rightarrow C_{pp}$ that maps a message to the commitment space C_{pp} using the random space given a message space M_{pp} and randomness space R_{pp}. For a message $x \in M_{pp}$, the algorithm samples $r_x \xleftarrow{R} R_{pp}$ and computes $c_x = Com_{pp}(x, r_x)$.*

When the context is clear, we drop the subscript and write $\mathsf{Com}_{\mathsf{pp}}$ as Com. We will focus on the Pedersen Commitment scheme [2], which is a kind of homomorphic commitment schemes defined as below:

Definition 3.2 (Homomorphic commitments)**.** *A homomorphic commitment scheme is a non-interactive commitment scheme such that M_{pp} and R_{pp} are fields (with $+, \times$) and C_{pp} is an Abelian group with the $\otimes$ operator, such that for all $x_1, x_2 \in M_{pp}$ and $r_1, r_2 \in \mathcal{R}_{pp}$ we have:*

$$Com(x_1, r_1) \otimes Com(x_2, r_2) = Com(x_1 + x_2, r_1 + r_2).$$

Informally, homomorphic commitment schemes have the following properties. The readers can refer to Ref. [1] for formal properties.

- **Hiding:** A commitment c_x reveals no information on x, r_x to a computationally bounded adversary.
- **Binding:** Given a commitment c_x to x using r_x, there is no efficient algorithm that can find $x' \neq x$ and r'_x such that $\mathsf{Com}(x', r'_x) = c_x$.
- **Zero Knowledge OR Opening:** Given c_x, the committing party can prove to a polynomial-time verifier that c_x is a commitment to either 1 or 0 without revealing which one it is. We denote such a proof as Π_{OR} and say it securely computes the oracle O_{OR}, which returns true if $c_x \in L_{\mathsf{Bit}}$:

$$L_{\mathsf{Bit}} = \{c_x : \exists x \in \{0, 1\} \wedge r_x \in \mathbb{Z}_q \wedge c_x = \mathsf{Com}(x, r_x)\}.$$

3.2 Differential privacy

We will consider computational differential privacy (DP) [3] in this paper, which is formally defined as follows:

Definition 3.3 (Computational DP [3])**.** *We consider $\kappa, n \in \mathbb{N}$, $\varepsilon \geq 0$, $\delta(\kappa) \leq \kappa^{-\omega(1)}$ a negligible function, and $\mathcal{M} = \{\mathcal{M}_\kappa : \mathcal{X}_\kappa^n \rightarrow \mathcal{Y}_\kappa\}_{\kappa \in \mathbb{N}}$ a family of randomized algorithms, where $\mathcal{X}_\kappa$ and $\mathcal{Y}_\kappa$ can be represented by $\mathrm{poly}(\kappa)$-bit strings. We say that $\mathcal{M}$ is computationally ε-differentially private if for every non-uniform PPT distinguisher D, every query $Q \in \mathcal{Q}$, and every pair of neighboring datasets $X \sim X' \in \mathcal{X}_\kappa^n$, and for all $T \subseteq \mathcal{Y}_\kappa$, we have:*

$$\Pr[D(\mathcal{M}_\kappa(X, Q)) \in T] \leq e^\varepsilon \Pr[D(\mathcal{M}_\kappa(X', Q)) \in T] + \delta(\kappa)$$

This is different from the standard (information-theoretic) (ε, δ)-DP introduced in the lectures [4], which provides guarantees against even an unbounded Turing machine. This is because information-theoretic verifiable DP is shown to be impossible in Ref. [1]: if both the prover and verifier are computationally unbounded, then statistical zero knowledge and unconditional soundness cannot hold simultaneously.

To achieve DP, we will use the binomial mechanism for a possible biased $p \neq 1/2$, which is a more generalized version of Lemma 2.7 in Ref. [1]:

Lemma 3.4 (Binomial Mechanism)**.** *Let $X = (x_1, \ldots, x_n) \in \mathbb{Z}_q^n$ and define the counting query $Q(X) = \sum_{i=1}^n x_i$. Fix $n_b > 0$, $0 < p < 1$, and $0 < \delta < 1$. Let $Z \sim \text{Binomial}(n_b, p)$. Then the mechanism $\mathcal{M}(X) = Q(X) + Z$ is (ε, δ)-differentially private with $\delta = o(1/n_b)$ and*

$$\varepsilon = 10\sqrt{\frac{1}{n_b p(1-p)} \ln\left(\frac{2}{\delta}\right)}.$$

The proof of Lemma 3.4 can be found in Ref. [5].

4 Verifiable DP using unbiased binomial mechanism

In this section, we provide the formal definition of verifiable DP proposed in Ref. [1]. We will then recap the concrete instantiations of verifiable DP in that work by computing DP counting queries (histograms) in the trusted curator and client-server multiparty settings using *unbiased* binomial mechanism at $p = 1/2$ and Pedersen Commitment scheme.

4.1 Verifiable DP

We now define protocols for verifiable differential privacy (DP) in the multi-party computation (MPC) setting. We also explain how this model simplifies to the trusted curator case. Let $\mathcal{M}_Q$ be a (possibly computational) differentially private mechanism for a query $Q \in \mathcal{Q}$. The protocol involves n clients with private inputs, a verifier Vfr, and K provers $(\mathsf{Pv}_1, \ldots, \mathsf{Pv}_K)$. Each party follows a next-message function that determines their communication based on their input, prior messages, and private randomness.

The protocol consists of three phases:

1. **Setup (Setup):** All parties agree on public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$ for security parameter κ. Each prover Pv_k receives shares of the client inputs, denoted by $\vec{X}_k = (\llbracket x_1 \rrbracket_k, \ldots, \llbracket x_n \rrbracket_k)$. The verifier receives commitments to these shares. All verifier messages are publicly visible.

2. **Verification (Verify):** If the query Q restricts the inputs to a subset $L \subseteq \mathcal{X}$ (e.g., requiring inputs to be binary), clients must prove in zero knowledge that their input satisfies this condition. Clients failing the verification are excluded.

3. **Computation (Π):** Once verification is complete, clients do not participate further. The remaining protocol Π is an interactive computation between the provers and the verifier. Over several rounds, the provers produce an output $\vec{y}$, and the verifier decides whether to accept or reject the result, based on its view and the public parameters. Let z be the verifier's auxiliary input, and $\vec{r}_v$ and $\vec{r}_{\mathsf{Pv}}$ are the verifier's and the provers' intrinsic randomness. Then the verifier's decision is $\mathsf{out}[\mathsf{Vfr}(\mathsf{pp}, \vec{r}_v, z), \vec{y}, \vec{\mathsf{Pv}}(\mathsf{pp}, \vec{r}_{\mathsf{Pv}})] \in \{0, 1\}$.

In the trusted curator model (a special case with $K = 1$), a single server sees all client data in plaintext and plays the role of the sole prover. The main difference is that, in MPC, inputs are secret-shared and no single prover sees the full data.

Now, we are ready to provide the full definition for verifiable DP in an informal and intuitive fashion:

Definition 4.1 (Verifiable Differential Privacy)**.** *A protocol* Π *for computing a DP mechanism* $\mathcal{M}$ *is said to be verifiable if, for any* n *clients and* $K \geq 1$ *provers, there exist negligible* ($n^{-\omega(1)}$ *scaling) functions* δ_c *and* δ_s *such that:*

1. ***Completeness:*** *If all parties are honest and follow the protocol, then the verifier accepts the output with probability at least* $1 - \delta_c$.

2. ***Soundness:*** *If a subset of provers behave maliciously, then any incorrect output (i.e., one not distributed as* $\mathcal{M}(X, Q)$*) is accepted with probability at most* δ_s.

3. ***Zero Knowledge:*** *For any subset of corrupted provers and a possibly corrupted verifier, there exists a simulator that can generate a view of the protocol indistinguishable from the real one, using only the true output and the corrupted parties' inputs and randomness. This ensures no extra information about client data is leaked.*

4.2 The protocol for MPC model using unbiased binomial mechanism

We now recap the protocol in Ref. [1], which describes how to compute counting queries verifiably with differential privacy in both the single curator and client-server MPC models using an unbiased binomial mechanism at $p = 1/2$. Here, Biswas and Cormode regard the trusted curator model to be a special instantiation of the general MPC at $K = 1$.

Let $\mathcal{X} = \mathbb{Z}_q = \mathcal{Y}$, where $\mathbb{Z}_q$ is a finite field of prime order q. Each client holds a private input $x_i \in \mathbb{Z}_q$ for $i \in [n]$, and the goal is to compute the total sum $Q(X) = \sum_{i=1}^{n} x_i$ in a differentially private and verifiable way. Each input is split into K additive secret shares, with prover Pv_k receiving share $[\![x_i]\!]_k$ of client i's input. We assume the field size q is large compared to the number of clients ($n \ll q$), and we define the security parameter as $\kappa = \lfloor \log_2 q \rfloor$.

We now describe the ideal functionality $\mathcal{M}_{\text{Bin}}$ that the protocol aims to securely realize:

1. $\mathcal{M}_{\text{Bin}}$ takes in public privacy parameters ε and δ, and computes the number of random bits n_b needed to sample unbiased binomial noise, as determined by Lemma 3.4 at $p = 1/2$.

2. Each prover Pv_k computes a local aggregate $X_k = \sum_{i=1}^{n} [\![x_i]\!]_k$ based on the shares it received. The prover then sends X_k to the ideal functionality. If the prover is malicious, it may send an arbitrary value.

3. For each prover's submitted value X_k, the ideal functionality samples independent unbiased binomial noise $\Delta_k \sim \text{Binomial}(n_b, 1/2)$. It then computes the noisy output as:

$$y = \sum_{k=1}^{K} (X_k + \Delta_k) \tag{1}$$

4. The ideal functionality sends each prover Pv_k its noise share Δ_k and the final result y. Each prover must respond with a CONTINUE signal to proceed. The functionality only continues once all provers have responded.

5. Once all provers have confirmed, $\mathcal{M}_{\text{Bin}}$ sends the final output y to the verifier Vfr. If any prover fails to send the CONTINUE message, the protocol halts and no output is given.

To sample unbiased coins publicly, we have to employ the Morra[1] oracle O_{morra}. We assume that both the provers and the verifier have access to oracles O_{morra} and O_{OR}. The full protocol is given in detail with line references to Figure 1.

1. **Line 1:** The verifier and prover agree on common public parameters pp, which include a cyclic group $\mathbb{G}_q$, generators for the Pedersen commitment scheme, the input and randomness spaces $\mathcal{M}_{\text{pp}} = \mathcal{R}_{\text{pp}} = \mathbb{Z}_q$, commitment space $\mathcal{C}_{\text{pp}} = \mathbb{G}_q$, and a formulation of $\mathcal{M}_{\text{Bin}}$ in Eq. (1).

[1] https://en.wikipedia.org/wiki/Morra_(game). See Algorithm 1 of Ref. [1] for the algorithmic implementation.

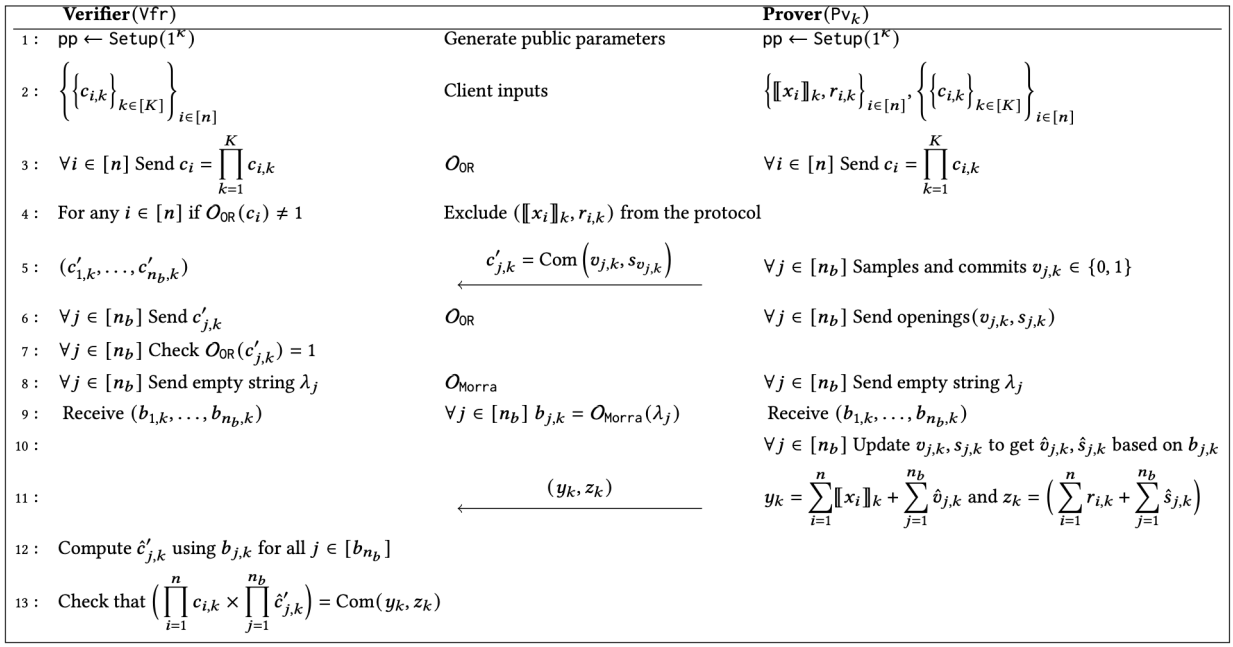

Figure 1: Compact description of the interactive protocol for K provers and the verifier computing $\mathcal{M}_{\text{Bin}}$.

2. **Line 2:** For prover Pv_k, client i sends the pair $(\llbracket x_i \rrbracket_k, r_{i,k})$ and broadcasts the corresponding commitment $c_{i,k} = \mathsf{Com}(\llbracket x_i \rrbracket_k, r_{i,k})$.

3. **Lines 3–4:** The verifier and prover compute the combined commitment $c_i = \prod_{k=1}^{K} c_{i,k}$ and submit it to $\mathcal{O}_{\text{OR}}$. The client reveals the opening $(x_i, r_i = \sum_k r_{i,k})$ to prove $x_i \in \{0, 1\}$. If this test fails, the client is excluded from the protocol.

4. **Line 5:** Each prover samples n_b private *unbiased* random bits $v_{j,k} \in \{0, 1\}$ for $j \in [n_b]$, creates commitments $c'_{j,k} = \mathsf{Com}(v_{j,k}, s_{j,k})$, and sends it to the verifier.

5. **Lines 6–7:** The verifier checks that each $c'_{j,k}$ commits to a valid bit using the oracle $\mathcal{O}_{\text{OR}}$. If any check fails, the prover is flagged as dishonest and the protocol is aborted.

6. **Lines 8-9:** All parties use the public coin oracle $\mathcal{O}_{\text{morra}}$ to sample n_b public unbiased bits $b_{j,k}$.

7. **Line 10:** For each index j, the prover computes:

$$\hat{v}_{j,k} = \begin{cases} 1 - v_{j,k}, & \text{if } b_{j,k} = 1 \\ v_{j,k}, & \text{if } b_{j,k} = 0 \end{cases} \quad \text{and} \quad \hat{s}_{j,k} = \begin{cases} 1 - s_{j,k}, & \text{if } b_{j,k} = 1 \\ s_{j,k}, & \text{if } b_{j,k} = 0. \end{cases}$$

8. **Line 11:** Each prover computes the following:

$$y_k = \sum_{i=1}^{n} \llbracket x_i \rrbracket_k + \sum_{j=1}^{n_b} \hat{v}_{j,k}, \quad z_k = \sum_{i=1}^{n} r_{i,k} + \sum_{j=1}^{n_b} \hat{s}_{j,k}$$

9. **Line 12:** The verifier, using the public bits $b_{j,k}$, transforms the commitments $c'_{j,k}$ accordingly. If $b_{j,k} = 1$, it computes $\hat{c}'_{j,k} = \mathsf{Com}(1, 1) \cdot (c'_{j,k})^{-1}$. Otherwise, $\hat{c}'_{j,k} = c'_{j,k}$.

10. **Line 13:** The verifier checks that the combined commitments match the prover's claimed output: $\left(\prod_{i=1}^{n} c_{i,k} \cdot \prod_{j=1}^{n_b} \hat{c}'_{j,k}\right) = \mathsf{Com}(y_k, z_k)$.

This protocol achieves the desired verifiable DP by the following theorem.

Theorem 4.2 (Theorem 4.1 of Ref. [1])**.** *Let $X = (x_1, \ldots, x_n)$ be the client input. Let $\mathcal{M}_{Bin}$ and $\mathcal{O} = (\mathcal{O}_{morra}, \mathcal{O}_{OR})$ be defined as above. Then the protocol Π_{Bin} in Figure 1 is a verifiable differentially private argument satisfying Definition 4.1.*

5 Verifiable DP using biased binomial mechanism

In this section, we provide our construction of Verifiable DP counting queries (histograms) in the single trusted curator model using the biased binomial mechanism. Here, we do not use the assumptions of

Biased bit flipping algorithm. To begin with, we provide a protocol to simulate the noise of one Bern(p) trial for general $p \in \mathbb{Q}$ in Algorithm 1.

Algorithm 1: Biased bit flipping.

Input: Biased probability p.
Output: A biased random coin.

1: Publicly represent p as a rational number $\frac{k}{m}$. k and m are publicly known.
2: The prover allocates a m bit vector $(v_1, \ldots, v_m)$ and randomly samples k of them. Set those k bits equal to 1 and the rest equal to 0.
3: The prover generates Pedersen commitments for the m bits $C_i = \mathsf{Com}(v_i; r_i)$, and sends all C_i to the verifier. Note that these values are subject to the binding property of Pedersen commitments.
4: The verifier uses Σ-OR [6] on each of the C_i to verify that each bit is either 0 or 1.
5: The verifier computes $C_{\text{sum}} := \prod_{i=1}^{m} C_i$ using the m commitments received. The prover reveals the aggregate randomness $r_\Sigma := \sum_i r_i$ which the verifier uses with publicly known k to compute $\mathsf{Com}(k; r_\Sigma)$. The verifier accepts if and only if $C_{\text{sum}} = \mathsf{Com}(k; r_\Sigma)$.
6: The verifier randomly samples an index $j \in \{1, .., m\}$. The bit at that index is 1 with probability $\frac{k}{m} = p$, and 0 with probability $1 - p$ because it was sampled uniformly at random.
7: The verifier already has the commitment of the bit at the j-th index, which they can use to do other checks.

Now, we show why Algorithm 1 prevents cheating as long as at least one prover and verifier is honest. Firstly, note that because of the initial commits of everything, the prover cannot lie about what the j-th bit was after the verifier picks it. As all bits were initially committed, by the binding property, all the bits are fixed. Moreover, the verifier multiplies the commitments to obtain C_{sum} and uses the released r_Σ to check $C_{\text{sum}} = \mathsf{Com}(k; r_\Sigma)$. Since the correct k is known by the verifier, then by the homomorphic binding property of Pedersen commitments, any discrepancy between the correct sum k and the committed bits would be caught immediately, thereby cryptographically tying k to that bit vector. In addition, the verifier already has the commits of all the m bits, so it would immediately be caught if the verifier tried to change things in post because the product of the commitments would not be equal. As long as at least one person is honestly randomly sampling, the probability is k/m as the probability of the j-th index being heads hinges on the fact that the j-th index was randomly sampled. This could come about in two ways. If the prover is cheating and they try to fix the k 1's instead of randomly sampling them from the m bits, we know the verifier still independently took a random sample of j, so the probability is

guaranteed. If the verifier is cheating by not randomly picking j, as long as the prover independently randomly picked the k indexes, this is a random permutation and the probability is guaranteed. If both are telling the truth, this is certainly an honest random pick of the j-th index, and the probability is guaranteed.

Alternatively, we may follow Ref. [7], representing the probability p as its finite rational bitwise decimal expansion, and then using a fair coin flipping oracle (such as in Ref. [1]). Intuitively we can flip unbiased coins independently until the first head appears at some j-th flip. In the protocol, we would ensure that p has a finite binary expansion, and flip all the bits in order to not leak any information about the exact outcome of the coin flip. Then, we would go through each flip and mark the first occurrence of 1 at the j-th flip. Then in the corresponding binary decimal p, we also check the j-th digit place, and if it is 1, we simulate the biased coin as heads, otherwise tails.

The protocol. Now, we are ready to present our protocol based on this biased bit flipping algorithm. We focus on the single curator model where we only have one prover Pv and one verifier Vfr.

Algorithm 2: Verifiable differential private counting based on biased binomial mechanism

1: Vfr and Pv agree on the shared parameters. For the Pedersen commitments, these are G_q, g, and h.
2: Pv sends the commitments of each x_i with their own randomness as $\mathrm{Com}(x_i, r_i)$ to Vfr.
3: Using a Σ-OR protocol, we check that each x_i is a bit.
4: Pv and Vfr sample the biased coins using Algorithm 1. Each time Algorithm 1 is run, the prover knows the value of the chosen bit $v \in \{0, 1\}$ that was equal to 1 with probability p for rational p, which they initially committed as $c_v = \mathrm{Com}(v, s)$. Meanwhile, the verifier has the commitment of this bit v, c_v. Algorithm 1 is run n_b times; the Binomial noise comes from summing all v's: $\sum_{j=1}^{n_b} v_j$. The commitment of this sum can be calculated by the verifier as $\prod_{j=1}^{n_b} c_{v_j}$.
5: Pv releases the raw noisy count $x_i + \sum_{j=1}^{n_b} v_j$, as well as the sum of all the noise added to the Pedersen commitments: $\sum_{i=1}^{n} r_i + \sum_{j=1}^{n_b} s_j$, where $\sum_{j=1}^{n_b} s_j$ is the sum of the noise used to commit the chosen v_j from each of the n_b runs of Algorithm 1.
6: The remaining part follows the same steps as lines 12–13 in Figure 1.

Regarding this protocol, we can show the following theorem, indicating that our protocol satisfies verifiable DP.

Theorem 5.1. *Let $X = (x_1, ..., x_n)$ be the client input. The above protocol is verifiable differential private and outputs $y = Q(X) + \Delta_k$ in the single trusted curator model, where $\Delta_k \sim \mathrm{Binomial}(n_b, p)$ for any p and $Q(X) = \sum_{i=1}^{n} x_i$.*

6 Randomized Response Protocol

6.1 Notation

1. $p = \frac{m}{2^k}$ with $1 \leq m \leq 2^k$ – we wish to express p as a rational number. $k \geq 1$ is the amount of public/private coins.

2. Private coins $v_1, \ldots, v_k \sim \mathrm{Bern}(1/2)$ and public coins $b_1, \ldots, b_k \sim \mathrm{Bern}(1/2)$ (given by $\mathcal{O}_{\mathrm{Morra}}$ from [1]). Then, let $d_i := v_i \oplus b_i$ be the XOR-ed coins which are still distributed $\mathrm{Bern}(\frac{1}{2})$ as long as at least the verifier or prover is honest.

3. Public statement $y = \big(C_x, C_{v_1}, \ldots, C_{v_k}, b_1, \ldots, b_k, o\big)$, witness $\omega = (x, v_1, \ldots, v_k, r_0, \ldots, r_k)$.

6.2 Toy Example ($p = \frac{1}{4} \implies k$=2, m=1)

Here we invoke a zero-knowledge proof of knowledge for the NP-relation:
$R = \left\{(y;\omega) \mid y = \left(C_x, C_{v_1}, \ldots, C_{v_k}, b_1, \ldots, b_k, o\right), \omega = (x, v_1, \ldots, v_k, r_0, \ldots, r_k) \text{ satisfies } C_x = \mathsf{Com}\left(x; r_0\right), C_{v_i} = \mathsf{Com}\left(v_i; r_i\right) \wedge v_i \in \{0,1\}(\forall i) \wedge o = x \oplus [U < m]\right\}$.

Define the commitments and check:

$$
\begin{aligned}
&C_x = \mathsf{Com}(x; r_0) \\
&C_{v_1} = \mathsf{Com}(v_1; r_1) \\
&C_{v_2} = \mathsf{Com}(v_2; r_2) \\
&d_1 = v_1 \oplus b_1, \quad d_2 = v_2 \oplus b_2 \\
&\text{if } (d_1, d_2) = (0,0) \text{ then } o = 1 - x \text{ else } o = x.
\end{aligned}
$$

Intuition: Let's say that for the coin flip, 1 is heads and 0 is tails. Then we know that if there was at least 1 honestly generated coin out of v_i or b_i, $d_i \sim$Bern($\frac{1}{2}$) since we are XOR-ing a true Bern($\frac{1}{2}$ bit with either a constant bit or another Bern($\frac{1}{2}$ bit. Then the event d_1, d_2 means that we flipped two tails. Out of all outcomes of 2 independent coin flips (HH, TH, HT, TT), two tails happens with probability $\frac{1}{4}$. Thus, $P(d_1, d_2) = \frac{1}{4} = p$, giving us our classic randomized–response mechanism from class. Now we can write a more formal algorithm for general p. We can also map the d_i into a draw from a Uniform distribution.

6.3 Randomized Response for flip probability p

Algorithm 3: Randomized Response with flip probability p

Input: A rational probability $p = m/2^k$, such that m and k are integers; client's private data $x \in \{0,1\}$
Output: A single biased bit o revealed to the verifier

1: Prover commits $C_x, C_{v_1}, \ldots, C_{v_k}$ and gives a Σ-OR proof each commitment is a bit.
2: Generate $b_1, \ldots, b_k \leftarrow \mathcal{O}_{\mathsf{Morra}}$.
3: Prover computes $d_i := v_i \oplus b_i$ and $U = \sum_{i=1}^{k} d_i 2^{i-1}$.
4: Output $o := x \oplus [U > m]$.
5: Prover proves in zero knowledge that Line 4 holds for the committed values.

6.4 Relaxed Assumptions

Note that we do not use the homomorphic property of commitments here because we only commit to individual bits $x, v_1, \ldots, v_k$; give Σ-OR proofs that each commitment opens to 0 or 1 ; and show in zero knowledge that the committed values satisfy the check $o = x \oplus [U < m]$. Therefore we never do arithmetic on commitments, thus it could be possible for us to relax the one-way homomorphic function assumption to simply a one-way function. However, note that this certainly depends on how Line 4 of Algorithm 3 is efficiently proven, as it is possible that Line 4 itself requires increased assumptions, etc.

Looking into these practical instantiations, the Σ–OR proofs and the final check $o = x \oplus [U < m]$ could potentially be realised with any general-purpose Boolean- or R1CS-based zero-knowledge system: Bulletproofs (dalek-bulletproofs), Halo 2 SNARK (libsnark, gnark, halo2), or STARK frameworks such as Winterfell. Based off the documentation, it is possible that these libraries provide gadgets that don't rely on any homomorphic property of the commitment itself.

6.5 Proof of correctness

Statement. We will prove that the algorithm outputs a bit o such that $\Pr[o \neq x] = p$ iff $\exists\, \omega = (x, v_1, \ldots, v_k, r_0, \ldots, r_k)$ satisfying Lines 1–4.

For the forward direction, assume an honest prover supplies a valid witness ω. We can show independence because every $d_i = v_i \oplus b_i$ contains the public fair bit b_i; hence $d_i \sim \text{Bern}(\frac{1}{2})$ and the vector $(d_1, \ldots, d_k)$ is uniform. Thus U is discrete uniform over $\{0, \ldots, 2^k - 1\}$. This means our bool $[U < m]$ equals 1 with probability $m/2^k = p$. Therefore Line 4 sets $o := x \oplus [U < m]$, so $\Pr[o \neq x] = p$, as desired,

For the backward direction, suppose the verifier receives a transcript for which the zero-knowledge proof accepts yet Lines 1–4 are *not* all true. The soundness of the proof system gives a contradiction, so acceptance implies a real witness ω. With that witness in place, the forward argument applies, yielding $\Pr[o \neq x] = p$. Thus both implications hold and we have proved that our algorithm correctly outputs a biased bit with flip probability exactly p if and only if the witness relations are satisfied. □

6.6 Alternative Randomized Response with probability p using Algorithm 1

We now give an informal sketch of how to obtain verifiable randomized response for general p using Algorithm 1. Observe that Algorithm 3 needs a single Bern (p) coin whose bias is hidden inside a zero-knowledge proof. Algorithm 1 already gives us a verifiable way to sample such a coin: the prover commits to an m-bit vector with exactly k ones $(p = k/m)$, proves this fact once, and then the verifier asks for one uniformly random position j; the bit at that position is 1 with probability $k/m = p$. Thus concretely we are invoking the special case of Algorithm 1 with $n = n_b = 1$

Therefore, to obtain Algorithm 3 we can simply replace Steps 1-4 of Algorithm 3 by one call to Algorithm 1, and reuse Algorithm 1's proof that the sampled bit is Bern(p). This would eliminate the need for the $k = \lceil \log_2(m) \rceil$ binary-expansion coins in Algorithm 3, trading them for the single "pick-a-random-index" trick of Algorithm 1. In both cases, the security guarantee comes from the same zero-knowledge proof subroutine.

7 *Accessible* implementations of verifiable DP

In order to implement the website mentioned above, Ari Biswas' code was modified to function through the methods of a single Rust struct, which contained the necessary data to successfully verify the correct usage of the binomial mechanism for differential privacy. This struct's methods were then wrapped in order to function as a server, and input and output JSONs suitable for TypeScript interfacing from a user-side frontend. This allowed the creation of a frontend site, accessible at zkdp.org, that teaches the user the basics of zero-knowledge DP proofs, and walks them through the process of actually proving their own data release of a counting query via the unbiased binomial mechanism. This backend supports both the unbiased and biased binomial mechanisms for DP counts, both through a very similar modular interface that makes it simple to understand and communicate with the backend.

In implementation, the backend uses the Ristretto method of generating finite-order fields of elliptic curves to represent commitments, and is thus far more cryptographically secure than a simple integer representation. Because of the computational costs of computing these commitments, much of the heavier computation was parallelized to allow faster data ingestion and processing. Commitments are flattened into scalars before being passed to the frontend, in order to allow displaying in understadable form.

On the frontend, the user can upload data in CSV form, select a column to release a count on, and select a threshold value to count lines by. After this, they are walked through the steps of committing bits, Sigma-OR proofs, playing Morra with their own private bits in order to generate provably random bits, computing their final DP count on the column and threshold, and releasing the final commitments that prove their implementation of the noise function for Differential Privacy was faithful.

By creating an accessible site such as this one, we hope that the fundamentals of zero knowledge DP proofs, as well as their usefulness, can be conveyed to a wider audience (much like DP Wizard and other such resources do for conventional DP). This will hopefully accelerate its adoption, and improve the transparency and verifiability of real-world DP implementations.

In the future, this Rust framework for verifiable differential privacy could also be formalized, and integrated into the OpenDP pipeline, allowing for easy widespread implementation of ZKDP proofs.[8]

All code can be accessed at the following GitHub page:
https://github.com/mwagner6/zkdp-exponential

References

[1] Ari Biswas and Graham Cormode. Interactive proofs for differentially private counting. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1919–1933, 2023.

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

[3] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. Computational differential privacy. In *Annual International Cryptology Conference*, pages 126–142. Springer, 2009.

[4] Salil Vadhan. The complexity of differential privacy. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pages 347–450, 2017.

[5] Badih Ghazi, Noah Golowich, Ravi Kumar, Rasmus Pagh, and Ameya Velingker. On the power of multiple anonymous messages: Frequency estimation and selection in the shuffle model of differential privacy. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 463–488. Springer, 2021.

[6] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Annual International Cryptology Conference*, pages 174–187. Springer, 1994.

[7] Donald Knuth. The complexity of nonuniform random number generation. *Algorithm and Complexity, New Directions and Results*, 1976.

[8] Marco Gaboardi, Michael Hay, and Salil Vadhan. A programming framework for opendp. *Manuscript, May*, 2020.