Malicious Security for Sparse Private Histograms

Lennart Braun[1]*, Adrià Gascón[2], Mariana Raykova[2], Phillipp Schoppmann[2], and Karn Seth[2]

[1] Aarhus University
[2] Google
braun@cs.au.dk,{adriag,marianar,schoppmann,karn}@google.com

Abstract. We present a construction for secure computation of differentially private sparse histograms that aggregates the inputs from a large number of clients. Each client contributes a value to the aggregate at a specific index. We focus on the case where the set of possible indices is superpolynomially large. Hence, the resulting histogram will be sparse, i.e., most entries will have the value zero.
Our construction relies on two non-colluding servers and provides security against malicious adversaries that may control one of the servers and any numbers of clients. It achieves communication and computation complexities linear in the input size, and achieves the optimal error $O\left(\frac{\log(1/\delta)}{\epsilon}\right)$, independent of the size of the domain of indices. We compute the communication cost of our protocol, showing its scalability. For a billion clients, the communication cost for each server is under 26 KiB per client.
Our paper solves an open problem of the work of Bell et al. (CCS'22) which presented a solution for the semi-honest setting while incurring sublinear overhead in its efficiency. We formalize a proof approach for proving malicious security in settings where the output and possible additional information revealed during the execution need to provide differential privacy.

* Part of the work was done while the author worked as research intern at Google.

Table of Contents

1 Introduction

Designing solutions that allow to learn the aggregate of inputs coming from many clients without learning individual data has been the objective of many works [Bon+17; CB17; Bon+21; Add+22; Bel+23; Li+23]. In order to protect individual data even in the setting of aggregation, there is a requirement that the output also provides differential privacy (DP) [Dwo+06]. Thus, it is important to obtain optimal parameters that allow to maximize the utility of the output while meeting the privacy bar for it.

Another challenge in many of the settings is that the domain of the possible input contributions is very large and the actual clients' inputs are very sparse within that domain. Thus solutions which require communication that is proportional to the domain size [Bon+17; CB17; Add+22; Bel+23; Li+23] are not applicable in these settings. While there have been constructions [BS15; ZXX16; Bas+20] that show how to use sketching techniques to manage the sparsity with logarithmic communication in the domain size D, they pay a price in terms of incurring $\log D$ factor in their privacy loss.

The work of Bell et al. [Bel+22] tackles the question of computing private distributed histogram over large sparse domains with communication proportional to the number of contributions and optimal differential privacy guarantees for the output. They present a construction which leverages two non-colluding servers to process the clients' contributions and provides security in the setting of semi-honest servers and clients. Their construction is a secure computation protocol that reveals to the servers some additional leakage about the inputs beyond the output private histogram, but this additional leakage is proven to be differentially private with respect to the clients' contributions.

The semi-honest requirement is needed for the security of that construction and, as the authors point out, relaxing it leads immediately to an attack where misbehaving clients colluding with one of the semi-honest servers can violate the privacy guarantees for the output. Their paper leaves as an open question how to achieve stronger security while preserving the optimal efficiency of their protocol and the optimal DP privacy parameters for the output.

In this work we resolve the open question of Bell et al. [Bel+22] and present a protocol for distributed sparse histograms that provides differential privacy for the clients' inputs. The computation is executed between two non-colluding servers. We prove security for our construction against malicious adversaries that control one of the servers and a subset of the clients. Similarly to Bell et al. [Bel+22] our protocol reveals additional information beyond the output to the servers which we prove is differentially private. The constructions achieve optimal error, independent of the domain size of the indices. It incurs only sublinear overhead compared with the semi-honest version of the protocol which means communication and computation linear in the inputs size.

Setting. There are N clients each of which has an input pair that consists of an index $u_i \in D$, where D is a large domain such that $|D| \gg N$, i.e., the contributions are sparse within the domain, and a value v_i. The desired output of the computation is an aggregate histogram of the clients' contributed values that is of the form $(i_j, V_j)_j$ where $V_j = \sum_{u_i = i_j} v_j$. However, this histogram should also provide differential privacy for the individual contributions.

The computation model allows two non-colluding servers to process the clients' contributions. The privacy guarantees require that the servers should learn only information that is differentially private with respect to the inputs. The ideal amount of information learnt by the servers will be just the differentially private histogram. However, the goal is to obtain a protocol that realizes the following efficiency, utility and security properties while minimizing the additional differentially private leakage about the inputs beyond the output. The protocol should require communication and cryptographic computation that scales linearly with the number of inputs and may depend only additively logarithmically on the domain size D. The error of the output $O\big(\frac{\log(1/\delta)}{\epsilon}\big)$ should match the optimal central DP accuracy. The privacy guarantees for the protocol should hold against a malicious adversary corrupting one of the servers and all but one of the clients.

Semi-honest construction. The construction of Bell et al. [Bel+22] achieves all of the above efficiency and accuracy goals, but only in the semi-honest setting where both servers and clients honestly follow their prescribed steps. In fact, relaxation of the semi-honest requirement allowing malicious clients leads directly to an attack on the privacy guarantees.

This work introduced several clever ideas of how to simulate optimal central DP mechanisms for sparse histogram in two party computation, which we also leverage in our construction. The DP mechanisms that achieve optimal error in the trusted curator model [Kor+09; BNS19] computes the non-private histogram $(i_j, V_j)_j$ from the inputs, then add DP noise to each of the non-zero values $V_j + \eta_j$ in this histogram and releases the indices and the noisy values when these values are above a threshold τ: $(i_j, V_j + \eta_j)_{\{j \mid V_j + \eta_j > \tau\}}$.

The challenge for emulating the above central DP mechanism in two-party computation is to do this without running time linear in the domain size and without revealing the indices where the noisy aggregates are below the threshold. Bell et al. [Bel+22] suggests a mechanism which leverages the multiplicity histogram of the input, which we can think of the histogram obtained by replacing the real indices with random values. It reveals the number j of indices that occur k times in the input for all values of k. They show an efficient protocol that allows the servers to compute a differentially private version of the multiplicity histogram using an oblivious PRF evaluation on the input indices, where one server holds the PRF key, and the other learns the OPRF output.

Working with the DP multiplicity histogram, which we also refer to as the *anonymous histogram* for the input since the indices are transformed to pseudorandom values, enables the servers to identify the contributions with the same index and whose values need to be added without revealing anything about distribution of the indices over the domain and while doing work only linear in the number of contributions. The remaining steps for adding noise and thresholding can also be executed efficiently.

Malicious security. Our new construction follows the idea of leveraging the multiplicity histogram to enable the input aggregation. However, this requires new techniques to enable the servers to compute the anonymous histogram from the clients' inputs.

The first challenge to resolve is the one that is also the reason for the attack possible in the presence of malicious clients. It stems from the fact that in the construction of Bell et al. [Bel+22] the clients are trusted to honestly perform the first step of the oblivious PRF evaluation, i.e., applying a hash function H which is modeled as random oracle, that transforms the indices into pseudorandom values. We replace the $H(u)^K$ PRF construction in that paper with the Dodis-Yampolskiy PRF [DY05] that allows oblivious evaluation between the two servers, where the client provides just an encryption of its index and is not trusted with any other steps. While there are previous works [Mia+20] that have shown maliciously secure oblivious PRF evaluation for the Dodis-Yampolskiy construction, our protocol achieves better efficiency by leveraging the HSM-CL encryption scheme [CLT18] which can be instantiated with plaintext size that matches the prime-order of the group used by the PRF. This modification on its own achieves a construction that provides security against semi-honest servers colluding with malicious clients.

In the semi-honest protocol [Bel+22] it suffices that only one of the servers generates the noise needed to achieve differential privacy for the multiplicity histogram. To achieve full malicious security against adversaries that control one server and any number of clients, we can no longer use this approach efficiently since it will require that the party adding the noise proves that it has correctly sampled from the appropriate DP distribution. We introduce completely different distributed noise generation protocols for the computation of the anonymous histogram which enables both servers to see the anonymous histogram and to perform the aggregation (without needing to provide a proof of correctness). Similarly, both servers can add noise to the aggregated values and do the thresholding in the clear.

As a result our malicious construction requires asymptotically the same amount of communication and computation as the semi-honest protocol.

Formal proof. The two-party computation construction here computes a functionality which outputs a differentially private sparse histogram. However, the protocol itself has additional leakage which is also differentially private. This opens up the question how to compose MPC and DP proofs to obtain a formal argument for the security guarantees of the construction. In the case of the semi-honest protocol [Bel+22], the security argument that the authors present has two steps: First, it computes the leakage from an honest execution and shows that it is differentially private. Second, it provides an MPC proof that the execution does not reveal anything more than the desired output plus the honest leakage.

The above argument is sufficient in the semi-honest case since we can think of the leakage and the output as something that can be generated by the ideal functionality as a single DP query on the underlying data even though these are revealed at different steps of the protocol. The important point is that semi-honest adversary cannot change the queries adaptively depending on what it may learn about the DP noise used in previous queries. In the malicious case this is no longer true since the adversary participated in the generation of the DP noise applied to queries.

We introduce a general methodology to argue security for MPC protocols with DP leakage in the malicious setting. In this case the interaction between the simulator and the ideal functionality can be viewed as a setting of interactive differential privacy queries [VZ23]. Thus, as a first step we need to prove that the view of the simulator in the interaction with the ideal functionality is indeed differentially private. To do this we show that the view of the simulator can be simulated using a non-interactive DP release from the database that is independent of the adversary's input. Once we can claim that the leakage in the ideal world is DP, we proceed with the simulation proof of our MPC protocol in the universal composability (UC) model using this simulator.

In our particular construction we first identify the maximum leakage that a malicious server can obtain from the database, which essentially is the private histogram from the semi-honest protocol [Bel+22] plus the additional leakage of the total number of frequency dummies generated by the servers and the total number of duplication. We show that this maintains DP properties for the input (see Theorem 3). We then construct a simulator for the MPC protocol who is interacting with the ideal functionality and we show: 1) that we can simulate the outputs that the simulator receives from the ideal functionality using the maximal leakage from above that we have proven DP (see Theorem 4), and 2) that the view of the adversary in the real world is indistinguishable from the view in the ideal world where it interacts with the simulator (see Theorem 5).

1.1 Technical Overview

Semi-honest Construction. We start with an overview of the technical details for the semi-honest construction of Bell et al. [Bel+22] which is the starting point for our construction. Understanding the approach of that work is essential in order to understand the challenges for the malicious protocol and the techniques we leverage to resolve them.

DP Multiplicity Histogram. As we discussed above a central contribution in the work of Bell et al. [Bel+22] is a protocol for two parties to efficiently learn a differentially private multiplicity histogram of the clients' indices $\mathcal{I} = (u_j)_{j\in[n]}$.

The starting observation one can use an oblivious PRF construction for the parties to learn a multiplicity histogram of $\mathcal{I}$. More concretely, the parties compute the multi-set $\mathcal{H}_{\mathcal{I}} := \{\{F_k(k, i) \mid i \in \mathcal{I}\}\}$ for a secret-shared key k, which reveals exactly the multiplicity histogram to a computationally bounded adversary. Therefore, we denote by $\mathcal{H}_{\mathcal{I}}$ the multiplicity histogram of $\mathcal{I}$. Concretely, $\mathcal{H}_{\mathcal{I}}$ is the mapping $[n] \to [n]$ such that $\mathcal{H}_{\mathcal{I}}[k] = j$ if and only if there are j distinct indices in $\mathcal{I}$ occurring exactly k times in $\mathcal{I}$. To see better why the PRF evaluation on the indices provides a multiplicity histogram is the fact that the information that the pseudorandom indices provide is just the cardinality of the contributions to different buckets (with no further information about the buckets) and thus we can count how many buckets have received k contributions.

However, the challenging task is to construct a differentially private approximation of $\mathcal{H}_{\mathcal{I}}$. For this purpose, Bell et al. [Bel+22] rely on two noising mechanisms that can be applied homomorphically on appropriately encrypted $\mathcal{I}$.

Consider an index $i \in \mathcal{I}$ with multiplicity k, i.e., appearing k times in $\mathcal{I}$. Bell et al. [Bel+22] describe two mechanisms that consist on generating fake client contributions. The first mechanism $\mathcal{M}_{\mathtt{freq}}$ only works under the assumption that $k < T$, for a fixed threshold $T > 0$, while the second one, denoted $\mathcal{M}_{\mathtt{dup}}$, works for $k \geq T$. The solution then consists on composing these mechanisms sequentially on the input $\mathcal{I}$, i.e. $\mathcal{M}_{\mathtt{dup}}(\mathcal{M}_{\mathtt{freq}}(\mathcal{I}))$.

- Frequency dummies ($\mathcal{M}_{\mathtt{freq}}$): This mechanism consists on noising $\mathcal{H}_{\mathcal{I}}[j]$, for each $j < T$, by introducing $\mathsf{X} \cdot j$ copies of a fresh index f_j disjoint with the domain of $\mathcal{I}$. Here, X is a positive random variable appropriately chosen to provide DP, e.g., a shifted discrete Laplace. One can think of the "frequency dummies" as simulating additional fake clients. Note that the communication overhead of this mechanism is significant, as it inflates $|\mathcal{I}|$ by $O(T^2)$ additional inputs.
- Duplication dummies ($\mathcal{M}_{\mathtt{dup}}$): This mechanism noises directly the counts in the buckets of $\mathcal{H}_{\mathcal{I}}$ that have more than T contributions by creating a number of copies each encrypted following an appropriately parameterized Negative Binomial distribution. Note that the duplication can be applied on encrypted entries without any knowledge of their destination bucket in the histogram, by copying and rerandomizing the ciphertexts. Duplication is applied on both the real clients' contributions as well as the frequency dummies.

In the construction by Bell et al. [Bel+22] one of the servers receives all encrypted inputs from the clients, generates the dummy items as above, shuffles real inputs and dummies and forwards them to the second server. All dummy items have contribution values of 0 so that they will not affect the actual aggregates that will be revealed. In this step the first server also applies its partial PRF evaluation as we discuss in more detail next. The second server who receives all shuffled real and dummy items can learn the total number of dummies that are added by virtue of knowing the input size.

Oblivious PRF constructions and Malicious Attack Bell et al. [Bel+22] compute an anonymous multiplicity histogram on the clients' input by transforming the indices in the input into pseudorandom indices by having the two servers jointly apply a PRF function. Specifically the construction uses the PRF $H(x)^K : \{0,1\}^* \to \mathbb{G}$ [JL09] where the key K is held by the first server. The distributed evaluation protocol is as follows. The client encrypts the hash of its index $H(u)$ under the public key of the second server and sends the ciphertext to the first server who holds the PRF key. The encryption is multiplicatively homomorphic which enables the fist server to apply the exponentiation by K on top of the encryption. The result can then be decrypted by the second server.

The security of the above construction fails in the presence of malicious clients. An adversary who controls malicious clients and the second server and wants to identify contributions for index u can instruct the clients to encrypt the values $H(u)$ and $H(u)^2$ instead of hashes of their indices. The oblivious PRF evaluation results in $H(u)^K$ and $H(u)^{2K}$. Now it is easy to see that these can be easily identified by a server who sees the pseudorandom indices just by looking for values such that one is the square of the other.

DP Sparse Histogram from DP Multiplicity Histogram. The transformation of the input to an anonymous multiplicity histogram in the protocol of Bell et al. [Bel+22] enables one of the servers who obtain the pseudorandom indices needed to group the client contributions whose values need to be added together. Those values are provided encrypted under additive homomorphic encryption, which allows adding encrypted values and rerandomizing them during the shuffling step needed to hide the dummy contributions added by the first server. The second server holds pairs $(H(u_i)^K, \mathsf{Enc}(v_i))$ and computes $\mathsf{Enc}(V_i) = \mathsf{Enc}(\sum_{u=u_i} v_i + \eta_u)$ where η_u is appropriately sampled DP noise.

The remaining steps are to compute a threshold for the value V_i which can be done by the first server who does not learn the PRF indices. For this purpose all v_i values are encrypted under a public key where the secret key is shared between the two servers. The second server that aggregates the values removes its part of the key and then passed the encrypted aggregates to the first server to complete the decryption.

Once the first server identifies the value $V_i > \tau$ bigger than the threshold τ for the DP protocol, the two servers jointly decrypt the value $\mathsf{Enc}(u)$ included in the input.

Malicious servers. In the following, we overview the main new insights and techniques that we introduce to obtain a construction with malicious security.

Different PRF with Efficient Oblivious Evaluation. As we discussed above, malicious client behavior leads to direct attack in the semi-honest protocol. To solve this issue we use a PRF that is more friendly to a malicious oblivious evaluation protocol. We use the Dodis-Yampolskiy PRF [DY05], which is of the form $\mathsf{F}^{\mathsf{DY}}(k, x) = g^{\frac{1}{k+x}}$ where $\mathbb{G} = \langle g \rangle$ is a group of prime-order q. The idea of the oblivious evaluation is to have one server homomorphically compute $\mathsf{Enc}(k + x)$, and then blind it as $\mathsf{Enc}(r \cdot (k + x))$. The second server can decrypt the blinded value and then compute $g^{\frac{1}{r\cdot(k+x)}}$. Now the first server removes the blinding factor to obtain the correct PRF evaluation $g^{\frac{1}{k+x}}$. This masking idea works well only when the plaintext space $\mathbb{F}_q$ for the encryption scheme and the group order $q = |\mathbb{G}|$ for the PRF match. We achieve this by using the HSM-CL encryption scheme [CLT18] and improving on previous work [Mia+20] that has constructed malicious oblivious evaluation for the Dodis-Yampolskiy PRF using Camenisch-Shoup (CL) encryption [CS03] which has a different plaintext domain.

Malicious DP Multiplicity Histogram Generation. In the semi-honest protocol of [Bel+22], one of the servers generates both the frequency and the duplication dummies. To preserve the same approach for the malicious setting, we would require the server to generate zero-knowledge proofs for correct sampling for DP noise distributions while hiding the sample. Moreover, the second server would need to prove in zero-knowledge that it performed the aggregation step correctly. We are not aware of any specialized ZK proof approaches for these functionalities. At the same time the functionality is quite complex even if we assume that the parties have generated jointly uniform randomness, so the use of generic approaches will result in high costs.

We adopt a different approach where both parties will add dummies in a distributed way such that they can both obtain the view of the anonymous histogram. We will use frequency and duplication dummies as in the semi-honest protocol to generate $\mathcal{M}^{\texttt{mal}}_{\texttt{freq}}(\mathcal{M}^{\texttt{mal}}_{\texttt{freq}}(\mathcal{I}))$ but the two mechanisms will be implemented differently.

- Frequency dummies ($\mathcal{M}^{\texttt{mal}}_{\texttt{freq}}$): each server will run independently the semi-honest protocol for frequency dummy generation $\mathcal{M}_{\texttt{freq}}(\mathcal{I})$ and the results of both executions will be used as frequency dummies. Since we are using a large domain for the indices of both the frequency dummies and the real input, the probability that the dummies of the malicious server collide with those of the honest server is negligible. Then, we can consider the malicious frequency dummies as additional clients' inputs and the security follows from the semi-honest protocol. This will result in a factor of two greater noise for the differential privacy property since the dummies from the honest server will have to be enough to guarantee differential privacy for the view of the malicious server.
- Duplication dummies ($\mathcal{M}^{\texttt{mal}}_{\texttt{freq}}$): Having each server generate its own set of duplication dummies does not work directly as in the case of frequency dummies. A misbehaving server could replicate a single input value in all of its duplication dummies instead of duplicating each real input and each frequency dummy with the appropriate probability. Generating a proof for correct duplication will require proving sampling correctly noise for each item, which, as we discussed above, is challenging to do efficiently.
  We adopt a different approach instead where the two servers shuffle the real contributions and frequency dummies and then jointly agree on a seed which determines how many times each item in the shuffled encrypted ciphertext will be duplicated. This reveals to each party the number of times each ciphertext (which is shuffled and cannot be linked to any real or dummy index) is duplicated. In order to make this leakage equivalent to revealing the total number of duplications, which is the case in the semi-honest protocol, we use the Bernoulli distribution to determine whether we duplicate each ciphertext. Thus, each ciphertexts is duplicated at most once and knowing which shuffled ciphertexts are duplicated will reveal only the total number of duplications. This results in DP noise per index sampled from a Binomial distribution $\mathsf{Bin}(n, p)$, in contrast to the semi-honest protocol which used negative binomial noise distribution. We show in Lemma 2 that this noise also provides differential privacy.

DP Sparse Histogram from DP Multiplicity Histogram. Our dummy generation mechanism allows both servers to see the differentially private multiplicity histogram. This enables them both to locally compute the encrypted aggregates of values per index $\sum_{u=u_i} \mathsf{Enc}(v_i)$ without learning the indices – avoiding costly zero-knowledge proofs of correct aggregation. Then, both servers can add independent noise samples η^1_u and η^2_u to the aggregate $\mathsf{Enc}(\sum_{u=u_i} v_i + \eta^1_u + \eta^2_u)$ before decrypting it jointly. For all values above the threshold τ, the two servers jointly decrypt $\mathsf{Enc}(u)$.

1.2 Related Work

Starting with the seminal Prio protocol [CB17], several works have explored two-server aggregation protocols for various aggregate functionalities. Prio+ [Add+22] extends the functionality to support inputs that are shared via boolean/XOR sharing, and Boneh et al. [Bon+19] and Davis et al. [Dav+23] improve the distributed zero-knowledge proofs that Prio needs to provide security agains malicious clients. Boneh et al. [Bon+21] introduce Poplar, a two-server aggregation protocol for a heavy hitters functionality, which can be viewed as a private histogram with a fixed threshold. More recently, both Davis et al. [Dav+23] and Mouris, Sarkar, and Tsoutsos [MST23] presented improved constructions for heavy hitters that build on Poplar. The main difference between all of the above works and ours is that they do not provide differential privacy for the output. And while they can be extended to provide DP, doing so while achieving a good privacy–utility trade-off has proven challenging for large, sparse domains. Bell et al. [Bel+22] provide the first two-server protocol with asymptotically optimal accuracy in this setting, and we follow their approach while strengthening the security guarantees.

Several large-scale deployments of two-server aggregation protocols have led to an increased industry interest in these technologies. Examples include Mozilla's Origin Telemetry [Moz22], Apple's and Google's Exposure Notification Privacy-preserving Analytics [AG21], and ISRG's Divvi Up service [Int22]. Driven by this industry interest, the IETF is working towards standardizing two-server aggregation protocols and their underlying cryptographic primitives [Bar+23; Geo+23].

An alternative threat model is followed by Secure Aggregation (SecAgg) [Bon+17]. Here, instead of two non-colluding servers, a single server is used, and privacy is guaranteed as long as a predefined fraction of clients remains honest. Several works [Rot+19; Bel+20; Bel+23; Li+23; Ma+23] have built on this paradigm, providing more efficient constructions, input validation, or better security and privacy guarantees. When considering sparse domains, works in the single-server setting are either using sketching [BS15; ZXX16; Bas+20], or use trie-based approaches [Zhu+20], requiring a privacy budget that scales with the domain size. To our knowledge, no work has considered private histograms with asymptotically optimal accuracy in the single-server setting. While our protocol can be trivially transferred to the single-server setting by replacing one of the two servers by an MPC committee of clients, doing so with satisfactory efficiency remains an open question.

2 Preliminaries

2.1 Notation

We use $[n]$ for the set $\{1, \ldots, n\}$, $[a, b]$ for $\{a, \ldots, b\}$, and $[a, b)$ for $[a, b-1]$. An ordered list is written as $[\![a, b, c]\!]$ and we write $\mathrm{Shuffle}(L)$ to denote that a list L is randomly permuted.

Let λ and σ denote the computational and a statistical security parameter, respectively. Let $q > 2^\lambda$ be a prime, and $\mathbb{F}_q$ the finite field of size q. For negative numbers, we use a centered representation $\mathbb{F}_q = \{-\lfloor q/2 \rfloor, \ldots, -1, 0, 1, \ldots, \lfloor q/2 \rfloor\}$.

We use prime-order groups as well as unknown order groups in our protocols: Let $\mathbb{G} = \langle g \rangle$ a cyclic group of order q where DDH is assumed to be hard (e.g., an elliptic curve group). We use the CL framework [CL15] for groups of unknown order with parameters generated as $\mathsf{pp}_{\mathsf{cl}} \leftarrow \mathsf{CLGen}(1^\lambda, q)$ with the same prime q. They define a class group $\widehat{G} \supset G = \langle g_{\mathsf{cl}} \rangle \simeq F \times G^q = \langle f \rangle \times \langle g_q \rangle$, where F is of order q with the discrete logarithm to base f efficiently computable, and G^q of unknown order. Let $\mathcal{D}_q$ induce an almost uniformly distribution in G^q.

2.2 Differential Privacy for Histograms

We are interested in histograms that aggregate datasats of up to N datapoints which are index-value pairs $x_i = (u_i, v_i) \in \mathcal{I} \times \mathcal{V}$. In practice the index set will be a finite field $\mathcal{I} = \mathbb{F}_q$ and the values are integers up to a sensitivity bound $\varDelta$ also represented as field elements, i.e., $\mathcal{V} = [0, \varDelta] \subseteq \mathbb{F}_q$. We write a dataset M as an ordered list/tuple $M = [\![x_1, \ldots, x_N]\!] \subseteq \bigcup_{n \in \mathbb{N}} (\mathcal{I} \times \mathcal{V})^n$. We define neighboring datasets M_0, M_1 to have the same size and differ in a single position x_{h}. We define a (non-interactive) *mechanism* $\mathcal{M}$ to be a randomized algorithm that takes a dataset M as input and releases some output.

Definition 1 (Differential Privacy [DR14]). *A mechanism $\mathcal{M}$ is (ε, δ)-differentially private if for all neighboring datasets M_0, M_1 in $\mathcal{M}$'s domain and for all $S \subset \mathrm{Range}(\mathcal{M})$: $\Pr[\mathcal{M}(M_0) \in S] \leq e^\varepsilon \cdot \Pr[\mathcal{M}(M_1) \in S] + \delta$.*

An *interactive* mechanism $\mathcal{M}$ is a randomized algorithm that takes a dataset as input and releases some output to the adversary (or analyst) $\mathcal{A}$. In contrast to before, we then can have several rounds of interaction, where $\mathcal{A}$ can adaptively send messages to $\mathcal{M}$, which releases more outputs depending on all the messages received so far. We denote the interaction between an interactive $\mathcal{M}$ and $\mathcal{A}$ as $\langle \mathcal{M}(M), \mathcal{A} \rangle$ and use $\mathsf{View}(\langle \mathcal{M}(M), \mathcal{A} \rangle)$ for the view of $\mathcal{A}$ in this interaction. This is a random variable depending on the dataset M as well as the random coins used by $\mathcal{M}$ and $\mathcal{A}$. We can also define differential privacy for such interactive mechanisms:

Definition 2 (Interactive Differential Privacy [VZ23]). *An* interactive *mechanism $\mathcal{M}$ is (ε, δ)-differentially private if for all neighboring datasets M_0, M_1 in $\mathcal{M}$'s domain, for all adversaries $\mathcal{A}$, and for all $S \subset \mathrm{Range}(\mathsf{View}(\langle \mathcal{M}(\cdot), \mathcal{A} \rangle))$:*

$$\Pr[\mathsf{View}(\langle \mathcal{M}(M_0), \mathcal{A} \rangle) \in S] \leq e^\varepsilon \cdot \Pr[\mathsf{View}(\langle \mathcal{M}(M_1), \mathcal{A} \rangle) \in S] + \delta.$$

In some of the proofs we rely on the the *tighter* definition of DP by means of hockey-stick divergences. The ε-hockey stick divergence between two distributions $\mathcal{U},\mathcal{U}'$ is defined as $d_\varepsilon(\mathcal{U}\|\mathcal{U}') := \sum_{\mathtt{support}(\mathcal{U})}[p_{\mathcal{U}}(x) - e^\varepsilon p_{\mathcal{U}'}(x)]_+$, where $p_{\mathcal{U}}(x)$ (resp. $p_{\mathcal{U}'}(x)$) denotes the probability mass of $\mathcal{U}$ (resp. $\mathcal{U}'$) at x, and $[y] = \max(y, 0)$. Distributions $\mathcal{U},\mathcal{U}'$ are (ε, δ)-close if $d_\varepsilon(\mathcal{U}\|\mathcal{U}') \le \delta$ and $d_\varepsilon(\mathcal{U}'\|\mathcal{U}) \le \delta$. Relating back to Definition 1, if distributions $\mathcal{M}(M_0)$ and $\mathcal{M}(M_1)$ are (ε, δ)-close for all neighboring datasets M_0, M_1, then $\mathcal{M}$ is differentially private.

Distributions We use the following probability distributions to sample noise for differential privacy throughout our paper. The *truncated, discrete Laplace distribution* with scale λ, denoted $\mathsf{TDLap}(\lambda, t)$, is the discrete distribution with support $[-t, t]$ and mass function $\propto \exp(-|x|/\lambda)$. As shown by Bell et al. [Bel+22], adding a noise sample from $\mathsf{TDLap}((, \lambda), t)$, with $\lambda = \Delta/\epsilon$, to the result of a sensitivity Δ query provides $(\epsilon, 2e^{-(t-\Delta)\epsilon/\Delta})$-DP. When noise values are required to be non-negative, we use the *truncated, shifted, discrete Laplace distribution*, denoted $\mathsf{TSDLap}(\lambda, t)$, with scale λ, support $[0, 2t]$, and mass function $\propto \exp(-|x-t|/\lambda)$. Analogous to before, adding a noise sample from $\mathsf{TSDLap}((, \lambda), t = \lceil \Delta + \Delta/\epsilon \log(2/\delta)\rceil)$ provides (ϵ, δ)-DP. For our duplication mechanism, we use the Bernoulli distribution $\mathsf{Ber}(p)$ with support $\{0, 1\}$ and mass function $p^x(1-p)^{1-x}$ to determine if a given (encrypted) index should be duplicated. Therefore, the number of duplicates after considering an set of n indices is binomially distributed. We denote the Binomial distribution as $\mathsf{Bin}(n, p)$, with support $[0, n]$ and mass function $(\binom{n}{k}p^x(1-p)^{1-x}$.

2.3 Pedersen Commitments

We use standard Pedersen commitments in the prime order group $\mathbb{G}$ with message space $\mathbb{F}_q$. The are information-theoretically hiding and computationally binding based on the discrete logarithm assumption. We use the notation $\mathsf{PedSetup}()$ to denote generation of the commitment public key $h_\mathsf{P} \in_R \mathbb{G}$, and we write $\mathsf{PedCommit}_{h_\mathsf{P}}(m; r)\colon \mathbb{F}_q \times \mathbb{F}_q \to \mathbb{G}$ to commit to a message $m \in \mathbb{F}_q$ by sampling $r \in_R \mathbb{F}_q$ and outputting $\mathsf{cm} := {h_\mathsf{P}}^m \cdot g^r$.

2.4 Encryption Schemes

In our histogram protocol, we use two different encryption schemes, ElGamal and HSM-CL, to encrypt values and indices, respectively. We use lifted ElGamal encryption [ElG84] over the group $\mathbb{G}$ where the message is encrypted in the exponent of g. The message space is $\mathbb{F}_q$, but only small messages can be decrypted efficiently. Instantiating $\mathbb{G}$ with elliptic curves yields relatively small ciphertexts. The second encryption scheme is HSM-CL [CLT18] with the same message space $\mathbb{F}_q$. The ciphertexts are larger compared to ElGamal, but HSM-CL has the advantage that *arbitrary* messages $m \in \mathbb{F}_q$ can be decrypted without size restriction.

The security proof of our histogram protocol uses that both encryption schemes admit so-called lossy public keys: A *lossy* public key is indistinguishable from a *normal* public keys, but all encryptions under this public key will statistically indistinguishable from encryptions of 0, i.e., they are independent of the encrypted message. These special keys are of the form $\mathsf{pk} = (\overline{\mathsf{pk}}, \mathsf{Enc}(\mathsf{pk}, b))$ with $b \in \{0, 1\}$, and encryption uses the homomorphic properties of the scheme to multiply the ciphertext from the public key (containing b) with a message m. In case of a normal key, we have $b = 1$, and for a lossy key $b = 0$ ensures that the resulting ciphertext does not contain any information about m.

We write the schemes as $\Pi_{\mathsf{HSM\text{-}CL}} = (\mathsf{KeyGen}^{\mathsf{cl}}_b, \mathsf{Enc}^{\mathsf{cl}}, \mathsf{Dec}^{\mathsf{cl}})$ and $\Pi_{\mathsf{ElGamal}} = (\mathsf{KeyGen}^{\mathsf{po}}_b, \mathsf{Enc}^{\mathsf{po}}, \mathsf{Dec}^{\mathsf{po}})$, respectively. To distinguish between data used by the two schemes, we use notation with a po-prefix for ElGamal in a <u>p</u>rime-<u>o</u>rder group (e.g., $\mathsf{posk}, \mathsf{popk}, \mathsf{poct}\text{-}m$ for secret keys, public keys, and ciphertexts containing some message m), and for HSM-<u>CL</u>, we analogously use a cl-prefix (e.g., $\mathsf{clsk}, \mathsf{clpk}, \mathsf{clct}\text{-}m$). We formally describe both encryption schemes in Figures 5 and 6 in Appendix A. Following the notation in [BDO23], we overload the operators $\cdot, +$ to describe homomorphic operations. Moreover, we use, e.g., $+^s_R$ to denote homomorphic addition followed by rerandomization with randomness s.

2.5 Oblivious Pseudorandom Functions

In our histogram protocol, we use a two-party oblivious PRF protocol to evaluate a PRF F on encrypted inputs $x_1, \ldots, x_n$ such that neither of the parties learns anything about the PRF key k or the inputs, but both parties obtain the outputs $\mathsf{F}(k, x_1), \ldots, \mathsf{F}(k, x_n)$. We make use of a PRF construction by Dodis and Yampolskiy [DY05], which especially friendly to oblivious evaluation (see e.g. [Mia+20]).

Definition 3 (Dodis-Yampolskiy PRF). *Let* $\{\mathsf{F}^{\mathsf{DY}} \colon \mathbb{F}_q \times \mathbb{F}_q \to \mathbb{G}_q\}_{q \text{ prime}}$ *be the family of functions where* $\mathbb{G}_q = \langle g \rangle$ *is a group of prime order* $q = \Omega(2^\lambda)$*, and* $\mathsf{F}^{\mathsf{DY}}(k, x) = g^{\frac{1}{k+x}}$*.*

Originally, Dodis and Yampolskiy proved standard PRF security for F^{DY}, but only for polynomially sized input spaces (i.e. $|\mathcal{X}| = \mathsf{poly}(\lambda)$) under the computational ℓ-DHI assumption (Definition 5 in Appendix B). In our case, the input space is large – it is the set of possible indices of the *sparse* histogram. Miao et al. [Mia+20] proved that with an exponentially large input space the Dodis-Yampolskiy PRF satisfies their weaker notion of a selectively secure PRF (Definition 7 in Appendix B) under the decisional ℓ-DHI assumption (Definition 5). In this work, we define the following notion of an *ℓ-non-adaptive pseudorandom function*, which is more convenient in our use case. The adversary $\mathcal{A}$ is allowed to make a fixed number ℓ of non-adaptive queries for some $\ell \in \mathbb{N}$. This is similar to the notion of [Mia+20], but now all outputs are either real or random, instead of only the last output.

Definition 4 (ℓ-Non-Adaptive PRF). *For an ensemble of PPT functions* $\{\mathsf{F}_\lambda \colon \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ *define the following game* $\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},b}(\lambda)$ *for a two-stage adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *and* $b \in \{\mathsf{real}, \mathsf{rand}\}$*: 1. Run* $(\mathsf{st}, (x_i)_{i\in[\ell]}) \leftarrow \mathcal{A}_1(1^\lambda)$*. 2. Sample* $k \in_R \mathcal{K}_\lambda$ *and* $H \in_R \mathsf{Funs}[\mathcal{X}_\lambda \to \mathcal{Y}_\lambda]$*. 3. Compute* $y_i^{(\mathsf{rand})} := H(x_i)$ *and* $y_i^{(\mathsf{real})} := \mathsf{F}_\lambda(k, x_i)$ *for* $i \in [\ell]$*. 4. Output* $\mathcal{A}_2(1^\lambda, \mathsf{st}, (y_i^{(b)})_{i\in[\ell]})$*.*

We say $\{\mathsf{F}_\lambda\}_\lambda$ *is a family of* ℓ-non-adaptive pseudorandom functions*, if for all PPT adversaries* $\mathcal{A}$ *and all large enough* $\lambda \in \mathbb{N}$*, we have*

$$\mathsf{Adv}^{\ell\text{-naPRF}}_{\mathcal{A}}(\lambda) := \left| \Pr\left[\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},\mathsf{real}}(\lambda)\right] - \Pr\left[\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},\mathsf{rand}}(\lambda)\right] \right| \le \mathsf{negl}(\lambda). \tag{1}$$

Definition 5 (Decisional ℓ-DHI Assumption (following [Mia+20])). *Let* $\mathbb{G} = \langle g \rangle$ *be a group of prime order* q*. We say that the* ℓ-DHI *assumption holds for* $\mathbb{G}$ *if for every PPT algorithm* $\mathcal{A}$ *we have*

$$\Big| \Pr\left[\mathcal{A}(1^\lambda, g, g^x, \dots, g^{x^\ell}, g^{\frac{1}{x}}) = 1 \mid x \in_R \mathbb{F}_q^*\right] - \\ \Pr\left[\mathcal{A}(1^\lambda, g, g^x, \dots, g^{x^\ell}, h) = 1 \mid x \in_R \mathbb{F}_q^*, h \in_R \mathbb{G}\right] \Big| \le \mathsf{negl}(\lambda),$$

where the probability is taken over the random choice of x *and the randomness used by* $\mathcal{A}$*.*

We prove that ℓ-non-adaptive security is implied by the notion of [Mia+20] (see Lemma 3 in Appendix B) and obtain the following theorem as corollary:

Theorem 1. *The Dodis-Yampolskiy PRF* F^{DY} *is* ℓ*-non-adaptively secure (Definition 4)) under the decisional* ℓ-DHI *assumption (Definition 5).*

2.6 Zero-Knowledge

To ensure security against a malicious adversary, our protocols make use of zero-knowledge proofs for various relations. To simplify notation, we use non-interactive proof systems in an abstract way. For a witness relation $\mathsf{R}_{\mathsf{Rel}}$, we write the corresponding proof system as $\Pi^{\mathsf{ZK}}_{\mathsf{Rel}}$. To simplify the notation, we assume that the relations are implicitly parameterized by the public parameters and public keys $\mathsf{clpk}, \mathsf{popk}$. Proving is written as $\pi \leftarrow \Pi^{\mathsf{ZK}}_{\mathsf{Rel}}.\mathsf{Prove}(x, w)$ for $(x; w) \in \mathsf{R}_{\mathsf{Rel}}$ and proofs are verified as $\{\top, \bot\} \leftarrow \Pi^{\mathsf{ZK}}.\mathsf{Verify}(\pi, x)$. Moreover, we use $\pi \leftarrow \Pi^{\mathsf{ZK}}.\mathsf{SimProve}(x)$ to simulate a proof. Note that this notation can hide, e.g., programming a random oracle when the Fiat-Shamir [FS87] transformation is used.

For most of the relations, the proofs can be instantiated using the canonical Σ protocols [Cra97] in prime-order groups and in class groups (cf. [BDO23]). In the following, we discuss the most important relations and the instantiations of the corresponding proofs. Further relations that are not used in the main body of this paper are deferred to Appendix D.

Proofs of Plaintext Knowledge To extract the corrupted parties' contributions in our security proofs, we need proof of plaintext knowledge for both encryption schemes. This also ensures that corrupted servers cannot create ciphertexts containing values that are related to the plaintext of honest users. Note that we only need to extract the message, but not the randomness. Hence, for HSM-CL, we can use the proof of plaintext knowledge from [BDO23], written as $\Pi^{\mathsf{ZK}}_{\mathsf{CL\text{-}PoPK}}$, and the standard Σ protocol for ElGamal $\Pi^{\mathsf{ZK}}_{\mathsf{PO\text{-}PoPK}}$, Note that due to our use of working/lossy public keys of the form $\mathsf{clpk} = (\overline{\mathsf{clpk}}, \mathsf{clpkct})$ (and analogous for ElGamal), the encryption relations are actually the relations for homomorphic multiplication with a constant followed by rerandomization.

$$\mathsf{R}_{\mathsf{CL\text{-}PoPK}} := \left\{\mathsf{clct}; (m, r) \;\middle|\; \mathsf{clct}_1 = \mathsf{clpkct}_1^m \cdot g_q^r \wedge \mathsf{clct}_2 = \mathsf{clpkct}_2^m \cdot (\overline{\mathsf{clpk}})^r\right\} \tag{2}$$

$$\mathsf{R}_{\mathsf{PO\text{-}PoPK}} := \left\{\mathsf{poct}; (m, r) \;\middle|\; \mathsf{poct}_1 = \mathsf{popkct}_1^m \cdot g^r \wedge \mathsf{poct}_2 = \mathsf{popkct}_2^m \cdot (\overline{\mathsf{popk}})^r\right\} \tag{3}$$

Range Proofs For the lifted ElGamal encryption, we additionally need to make sure that the encrypted values lie in a certain range $[a, b] \subseteq \mathbb{F}_q$. This is i.a. needed to ensure that decryption always succeeds. We can instantiate $\mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{Range}\text{-}[a,b]}$ for example with Bulletproofs [Bün+18].

$$\mathsf{R}^{[a,b]}_{\mathsf{Range}} := \left\{\mathsf{poct}; (m, r) \;\middle|\; (\mathsf{poct}; (m, r)) \in \mathsf{R}_{\mathsf{PO\text{-}PoPK}} \wedge m \in [a, b]\right\} \tag{4}$$

Client Inputs The clients need to prove with $\mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{Client}}$ that their contribution is well-formed, i.e., they send valid ciphertexts containing index and value in the appropriate ranges:

$$\begin{aligned}\mathsf{R}_{\mathsf{Client}} := \Big\{&(\mathsf{clct}, \mathsf{poct}); (u, r, v, s) \;\Big|\\ &(\mathsf{clct}; (u, r)) \in \mathsf{R}_{\mathsf{CL\text{-}PoPK}} \wedge (\mathsf{poct}; (v, s)) \in \mathsf{R}^{[0,\Delta]}_{\mathsf{Range}}\Big\}.\end{aligned} \tag{5}$$

Shuffle Proofs We repeatedly need to randomly shuffle lists of ciphertext pairs consisting of one ElGamal and one HSM-CL ciphertext. It is essential that after the shuffle each pair of input ciphertexts corresponds exactly to one pair of output ciphertexts while ensuring that the random permutation stays hidden. Hence, we rerandomize the permuted ciphertexts and then prove correctness of the whole operation in zero-knowledge. Moreover, for the security proof of our protocol, we will need to extract the permutation, but not necessarily the randomness used for rerandomization. We can use the protocols by Bayer and Groth [BG12] or by Hoffmann et al. [HKR19] to instantiate a shuffle proof $\mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{Shuffle}}$ for the following relation:

$$\begin{aligned}\mathsf{R}_{\mathsf{Shuffle}} := \Big\{&((\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i)_{i\in[n]}, (\mathsf{clct}\text{-}u'_i, \mathsf{clct}\text{-}v'_i)_{i\in[n]}); (\sigma, (r_i)_{i\in[n]}, (s_i)_{i\in[n]})\\ &\Big|\; ((\mathsf{clct}\text{-}u_i, \mathsf{clct}\text{-}u'_{\sigma(i)}); r_i) \in \mathsf{R}^{\mathsf{cl}}_{\mathsf{Randomize}} \text{ for } i \in [n]\\ &\wedge\; ((\mathsf{poct}\text{-}v_i, \mathsf{poct}\text{-}v'_{\sigma(i)}); s_i) \in \mathsf{R}^{\mathsf{po}}_{\mathsf{Randomize}} \text{ for } i \in [n]\Big\}.\end{aligned} \tag{6}$$

OPRF Subprotocol In our OPRF protocol $\mathit{\Pi}_{\mathsf{OPRF}}$, we use zero-knowledge proofs for the following two relations:

$$\begin{aligned}\mathsf{R}_{\mathsf{AddBlindCom}} := \Big\{&(\mathsf{clct}_k, \mathsf{clct}, \mathsf{clct}', \mathsf{cm}); (r, s, \rho)\\ &\Big|\; \mathsf{clct}' = (\mathsf{clct} + \mathsf{clct}_k) \cdot^s_R r \wedge \mathsf{cm} = \mathsf{PedCommit}(r; \rho)\Big\},\end{aligned} \tag{7}$$

$$\begin{aligned}\mathsf{R}_{\mathsf{ExpCom}} := \Big\{&(h, t, \mathsf{cm}); (r, \rho)\\ &\Big|\; t = h^r \wedge \mathsf{cm} = \mathsf{PedCommit}(r; \rho)\Big\}.\end{aligned} \tag{8}$$

2.7 Auxiliary Functionalities

We make use of some auxiliary functionalities, which are formally described in Appendix C. First, we assume a way for the servers to reliable broadcast messages to all clients and model this as functionality $\mathcal{F}_{\mathsf{SBCast}}$ (Figure 7). Then, we use a commitment functionality $\mathcal{F}_{\mathsf{Com}}$ (Figure 8), which has a simple instantiation in the random oracle model, and a functionality $\mathcal{F}_{\mathsf{Rand}}$ (Figure 9) that outputs random bit-strings with each bit Bernoulli-distributed.

Finally, we use $\mathcal{F}_{\mathsf{Enc}}$ (Figure 10), which models a two-party threshold encryption scheme for ElGamal and HSM-CL encryption (see Section 2.4). It has the methods KeyGen, CL-Decrypt, and PO-Decrypt to generate key pairs, and decrypt $\mathit{\Pi}_{\mathsf{HSM\text{-}CL}}$ and $\mathit{\Pi}_{\mathsf{ElGamal}}$ ciphertexts, respectively. Moreover, $\mathcal{F}_{\mathsf{Enc}}$ contains the methods CL-Random and CL-Constant to generate $\mathit{\Pi}_{\mathsf{HSM\text{-}CL}}$ ciphertexts with a random or publicly known value. PO-Zero is used to create an Π_{ElGamal} encryption of zero. Finally, DecExpInv decrypts given $\Pi_{\mathsf{HSM\text{-}CL}}$ ciphertexts clct-a_i and outputs $g^{\frac{1}{a_i}} \in \mathbb{G}$ while leaking a_i to one of the servers. We formally define $\mathcal{F}_{\mathsf{Enc}}$ in Appendix E which also contains an instantiation with the protocol Π_{Enc} and the corresponding security proof.

3 Differentially Private Sparse Histograms

In this section, we introduce the DP mechanism implemented by our protocol. It follows a similar structure as Bell et al. [Bel+22], but uses a different dummy distribution for duplication dummies. To simplify the description, we analyze our algorithms under add/remove DP. However, our final privacy theorem (Theorem 3) is stated in terms of standard substitution DP.

3.1 Mechanism

The starting point of our DP mechanism is the work of Bell et al. [Bel+22]. We define our variant $\mathcal{M}_{\mathsf{Hist}}$ in Figure 1. The formal definition of the noise generation algorithms is deferred to Appendix F.

The main difference with the work of Bell et al. is that we sample duplicate dummies in step 1(b) using the Bernoulli distribution, i.e., the mechanism tosses a coin with bias p to determine if a given index must be duplicate or not. In contrast, Bell et al. samples a Negative Binomial, which dictates *how many times* a given index must be duplicated. The latter is more efficient in terms of communication overhead for small ε, as the number of required dummies scales with $1/\varepsilon$, as opposed to the Bernoulli case, which scales with $1/\epsilon^2$. However, the advantage of the approach based on Bernoulli in our setting is that the servers can jointly toss coins as long as we can guarantee that they will have the right bias. This leads to an efficient protocol where servers can simply sample a seed to derive unbiased coins from which coins with the right bias can be securely obtained.

The mechanism consists on three noise addition stages. In the first one, so-called frequency dummies are added (step 1(a)). These are indices sampled from a set sampled from a set of indices disjoint with the input. The purpose of this mechanism is to provide DP for indices occurring less that a threshold T number of times. Next, the duplication dummies discussed above are generated. These are obtained by duplicating according to a Bernoulli distribution with parameter p the set of indices obtained so far (including the previously generated frequency dummies). Next, the multiplicity histogram $\mathcal{H}^A$ of the resulting set of indices is computed (step 3). $\mathcal{H}^A$ is outputted by the mechanism, along with both (a) the total number of frequency dummies added in Step 1(a), and (b) the total number of duplicating dummies added in Step 1(b). Showing that this triple $(|D_{IF}|, |D_{ID}|, \mathcal{H}^A)$ is differentially private is the main challenge in the proof. Finally, the third noise addition mechanism samples dummy buckets as a way to randomize an anonymous histogram with differentially private counts. Note that removing a user's input might alter the output of the corresponding anonymous histogram computation by either (i) reducing a count in a bucket by the sensitivity Δ or (ii) introducing a new bucket with value in $[\Delta]$. The sampling of dummy buckets accounts for (ii) by adding appropriately distributed buckets with all possible values $1, \ldots, \Delta$.

DP Mechanism $\mathcal{M}_{\mathsf{Hist}}$ for Sparse Histogram

Parameters: sensitivity Δ, size of the dataset N, noise parameters $(t_3, \lambda_3), p, (t_2, \lambda_2), (t_1, \lambda_1)$, threshold $\tau = \Delta + t_1 + 1$, threshold $T > 0$ for frequency and duplicate dummy generation, dummy domain $\mathcal{D}$ that is disjoint from the input domain.
Input: A dataset written as an ordered list $M = [\![(u_i, v_i)]\!]$ of length N, with $(u_i, v_i) \in \mathcal{I} \times [0, \Delta]$ for $i \in [N]$. Let $M_U := [\![u \mid (u, \_) \in M]\!]$ be the list of the indices in M.

1. Add dummy contributions that include an index, but have no value:
   (a) $D_{IF}, \_ \leftarrow \mathsf{SampleFrequencyDummies}(\mathcal{D}, T, \lambda_3, t_3, \mathsf{plain}, \perp)$, and
   (b) $D_{ID} \leftarrow \mathsf{SampleDuplicateDummies}(M_U \mid\mid D_{IF}, p)$.
   Set $M' := M \mid\mid [\![(u, 0) \mid u \in (D_{IF} \mid\mid D_{ID})]\!]$.
2. Define the following:
   (a) Let $U := \{u \mid (u, \_) \in M'\}$ be the set of occurring bucket indices, and let $B := |U|$ be the number of buckets.

(b) For $u \in U$, let $V_u := \{v \mid (u,v) \in M'\}$ be the multiset of values for the bucket with index u, and let $V := \biguplus_{u \in U} V_u$ be the multiset of overall occurring values.

3. Compute the anonymous histogram of M', i.e., the (sorted) list

$$\mathcal{H}^A := \llbracket n_u := |\{(x,y) \in M' \mid x = u\}| \mid u \in U \rrbracket. \tag{9}$$

4. Compute the (normal) histogram of M'[a] as sorted list:

$$\mathcal{H} := \left\llbracket (u,v) \;\middle|\; u \in U \wedge v := \textstyle\sum_{v_i \in V_u} v_i \right\rrbracket. \tag{10}$$

5. Add dummy buckets:
   (a) $D_B \leftarrow \mathsf{SampleDummyBuckets}(\Delta, \lambda_2, t_2, \mathsf{plain}, \perp)$
   (b) Set $\mathcal{H}' := \mathcal{H} \mathbin{||} \llbracket (\perp, v) \mid v \in D_B \rrbracket$ with $B' := |\mathcal{H}'|$.
6. Compute the private histogram:
   (a) Write $\llbracket (u_1, v_1), \ldots, (u_{B'}, v_{B'}) \rrbracket := \mathcal{H}'$.
   (b) Sample noise as $\zeta_i \leftarrow \mathsf{TDLap}(\lambda_1, t_1)$ for $i \in [B']$.
   (c) Compute the private histogram[b] as list:

$$\mathcal{H}^P := \llbracket (u_i, v_i + \zeta_i) \mid v_i + \zeta_i \geq \tau \rrbracket. \tag{11}$$

   (d) Compute the threshold leakage as a shuffled list:

$$V^{\perp} := \mathrm{Shuffle}(\llbracket v_i + \zeta_i \mid v_i + \zeta_i < \tau \rrbracket). \tag{12}$$

Output: Release the following: $(|D_{IF}|, |D_{ID}|, \mathcal{H}^A, |D_B|, V^{\perp}, \mathcal{H}^P)$.

[a] Except for potentially additional zero-values buckets, this coincides with the histogram of the original dataset M, since the dummy elements all have an associated value of zero.

[b] $\mathcal{H}^P$ will not contain any entries of the form $(\perp, v)$, since they never make it above the threshold τ.

Fig. 1: Description of the DP Mechanism implemented by our protocol.

Before we prove the fact that the mechanism is DP we introduce auxiliary lemmas regarding the frequency and duplication dummies. The following lemma states that adding dummies according to SampleFrequencyDummies provides DP for inputs with multiplicity below T. It is not hard to see by inspecting the definition of SampleFrequencyDummies that the expected number of dummies added by this mechanism is $O_{\epsilon,\delta}(T^2)$.

Lemma 1 (Frequency Dummies via Laplace [Bel+22]). *Let D be a multi-set of indices in $\mathcal{I}$, and let $T > 0$ be an integer threshold value. Let $i \in D$ be an index such that $|D|_i = k \leq T$, and let D' be the multi-set obtained by removing one copy of i from D. Let $\mathcal{M}(X)$ be $\mathsf{SampleFrequencyDummies}(\mathcal{D}, T, \lambda, t, \mathsf{plain}, \perp)$. Then, $\mathcal{M}(D)$ and $\mathcal{M}(D')$ are (ε, δ)-indistinguishable for $\lambda = \log(1/\delta)/\epsilon$ and $t = 1 + \log(1/\delta)/\epsilon$.*

The following theorem states that adding appropriately calibrated binomial noise provides differential privacy for sensitivity k queries. We use this result in the proof of the subsequent lemma, which shows that our duplication-based mechanism achieves differential privacy.

Theorem 2 ([Gha+21]). *For any $\varepsilon, \delta \in (0,1)$, and $k > 0$, let $n \geq \frac{90k^2 \ln(2/\delta)}{\varepsilon^2}$ and p be such that $\frac{90k^2 \ln(2/\delta)}{n\varepsilon^2} \leq p \leq 0.5$. Then, we have*

$$d_\varepsilon(\mathsf{Bin}(n,p) + k \| \mathsf{Bin}(n,p)) \leq \delta,$$
$$d_\varepsilon(\mathsf{Bin}(n,p) \| \mathsf{Bin}(n,p) + k) \leq \delta.$$

Lemma 2 (Duplicate Dummies via Binomial). *Let D be a multi-set of indices in $\mathcal{I}$, and let $T > 0$ be an integer threshold value. Let $i \in D$ be an index such that $|D|_i = k > T$, and let D' be the multi-set obtained by removing one copy of i from D. Let $\mathcal{M}(X)$ be a mechanism that replicates every value* *in a multi-set X with probability p. Then, $\mathcal{M}(D)$ and $\mathcal{M}(D')$ are (ε, δ)-indistinguishable, so long as $T \geq \frac{360 \ln(2/\delta)}{\varepsilon^2}$ and $\frac{360 \ln(2/\delta)}{T\varepsilon^2} \leq p \leq 0.5$.*

Proof. We first show that (i) $d_\varepsilon(\mathsf{Bin}(T+1,p)+1 \| \mathsf{Bin}(T,p)) \leq \delta$ and that (ii) $d_\varepsilon(\mathsf{Bin}(T,p) \| \mathsf{Bin}(T+1,p)+1) \leq \delta$, i.e., that adding up enough Bernoulli random variables with small enough bias hides the presence of a value in the sum, in the sense of (ε, δ)-closeness. That our duplication mechanism is DP follows quite directly from this result.

First, note that that $\mathsf{Bin}(T+1,p)+1$ can be written as a mixture of distributions $\mathsf{Bin}(T,p)+2$ and $\mathsf{Bin}(T,p)+1$, where the former is chosen with probability p. Therefore, we have $d_\varepsilon(\mathsf{Bin}(T+1,p)+1 \| \mathsf{Bin}(T,p)) \leq \max\{d_\varepsilon(\mathsf{Bin}(T,p)+2 \| \mathsf{Bin}(T,p)), d_\varepsilon(\mathsf{Bin}(T,p)+1 \| \mathsf{Bin}(T,p))\} \leq \delta$, where the first inequality follows from the definition of hockey stick divergence, and the last one follows from Theorem 2, with $k = 2$, and the choice of T and p in the statement of the lemma. The proof that $d_\varepsilon(\mathsf{Bin}(T,p) \| \mathsf{Bin}(T+1,p)+1) \leq \delta$ is analogous.

Having concluded (i) and (ii), the statement of the Lemma follows from a standard DP analysis. Note that the multiplicity histograms $\mathcal{H}_D$ and $\mathcal{H}_{D'}$ corresponding to $\mathcal{M}(D)$ and $\mathcal{M}(D')$, respectively, differ only in bucket k. Moreover $\mathcal{H}_D$ and $\mathcal{H}_{D'}$ are (ε, δ)-close if $\mathsf{Bin}(k+1,p)+1$ and $\mathsf{Bin}(k,p)$ are (ε, δ)-close, which follows from (i, ii) and the fact that $k > T$. ☐

It is worth remarking that the expect number of dummies added by duplication is $O_{\varepsilon,\delta}(n/T)$, in expectation. Therefore choosing $T = O(n^{1/3})$ yields a total number of dummies, including frequency dummies, that is sublinear in n.

Theorem 3. *Let ε, δ be privacy parameters. Let $\varepsilon_i = \varepsilon/3$ and $\delta_i = \delta/3$, for $i \in [3]$. The mechanism $\mathcal{M}_{\mathsf{Hist}}$ is (ε, δ)-differentially private for $T(\varepsilon_3/2)$ as required by Lemma 2, $p(\varepsilon_3/2, \delta_3/(1+e^{\varepsilon_3}))$ as required by Lemma 2, $t_3 = 1 + 2\log((1+e^{\varepsilon_3})/\delta_3)/\varepsilon_3$, $\lambda_3 = \log((1+e^{\varepsilon_3})/\delta_3)/\varepsilon_3$, $t_2 = 1 + \log(1/\delta_2)/\varepsilon_2$, $\lambda_2 = 2\log(2/\delta_2)/\varepsilon_2$, $t_1 = \Delta + \log(1/\delta_1)$, and $\lambda_1 = \Delta \log(2/\delta_1)/\varepsilon_1$.*

Proof. To prove that the output distributions $(|D_{IF}|, |D_{ID}|, \mathcal{H}^A, |D_B|, V^\perp, \mathcal{H}^P)$ under two neighboring datasets D_0, D_1 are (ϵ, δ)-close we first consider the tuple $(|D_{IF}|, |D_{ID}|, \mathcal{H}^A)$. Assume that D_1 is obtained from D_0 by removing an index-value pair whose index has multiplicity k in D_0. We distinguish to cases: $k \leq T$, and $k > T$. In the former case the fact that $(|D_{IF}|, |D_{ID}|, \mathcal{H}^A)$ is DP follows from Lemma 1, as the quantity $|D_{ID}|$ can be seen as post-processing of a differentially private value (recall that the mechanism also duplicates frequency dummies). Note that while Lemma 1, is stated in terms of removal DP, we set privacy budget according to the equivalence that if a function is removal-DP with parameters (ε, δ), then it is substitution-DP with parameters $2\varepsilon, ((1+e^\varepsilon)\delta)$. The case where $k > T$ follows from Lemma 2, under substitution DP, given the parameter choices for T, p in the statement of the theorem. Note that similarly as above the claim holds for substitution DP, and therefore the $\mathcal{H}^A$ is DP even when revealed along with $|D_{ID}|$. That the second part of the view, namely $|D_B|, V^\perp, \mathcal{H}^P$ is DP follows from a standard application of the Laplace mechanism and a stability argument for non-thresholded values given the choice of τ, just like in [Bel+22]. ☐

3.2 UC Ideal Functionality

Since we are about to construct an MPC protocol for computing sparse histograms and prove its security we need to specify exactly what security properties it should satisfy. We use the universal composibility model, so we do this by defining the UC ideal functionality $\mathcal{F}_{\mathsf{Hist}}$ in Figure 2.

The functionality $\mathcal{F}_{\mathsf{Hist}}$ is defined for N clients C_i that provide inputs (u_i, v_i) and two servers S_1 and S_2 who obtain a private histogram as output. The adversary (or environment) $\mathcal{Z}$ is able to statically and actively corrupt up to $N-1$ clients and one of the servers. To obtain a more efficient protocol, we allow a corrupted server to learn additional information and influence the computation of $\mathcal{F}_{\mathsf{Hist}}$. This is specified in $\mathcal{F}_{\mathsf{Hist}}$ by receiving influence from and sending leakage to the simulator (a.k.a. ideal adversary) $\mathcal{S}_{\mathsf{Hist}}$.

Since the simulator $\mathcal{S}_{\mathsf{Hist}}$ interacts with the functionality $\mathcal{F}_{\mathsf{Hist}}$ we can interpret the latter as an interactive mechanism in the context of differential privacy (see Section 2.2). While it has essentially the same goal as the non-interactive mechanism $\mathcal{M}_{\mathsf{Hist}}$ from Section 3.1, it is defined with the UC formalities in mind to enable a security proof of the corresponding MPC protocol. For example, the simulator $\mathcal{S}_{\mathsf{Hist}}$ is allowed to contribute its own noise to the noisy histograms, which corresponds to the noise of the corrupted party in our MPC protocol. In the following Section 3.3, we prove that the view of $\mathcal{S}_{\mathsf{Hist}}$ is differentially private despite the additional influence and leakage.

UC Ideal Functionality $\mathcal{F}_{\mathsf{Hist}}$ for Sparse Histograms

Parameters: sensitivity Δ, number of clients N, noise parameters, $(t_3, \lambda_3), p, (t_2, \lambda_2), (t_1, \lambda_1)$, threshold $\tau_{\mathcal{F}} = \Delta + 2 \cdot t_1 + 1$

Init On input (Init) from S_j, $j \in \{1, 2\}$, send $(\mathsf{InitReceived}, \mathsf{S}_j)$ to $\mathcal{S}_{\mathsf{Hist}}$ and ignore futher (Init) messages from S_j. Once both servers have sent (Init), initialize an empty list $M = [\![]\!]$ and send $(\mathsf{Initialized})$ to all C_i.

Client Input This method must be called after initialization is completed. On input $(\mathsf{Input}, u_i, v_i)$ from C_i, where $(u_i, v_i) \in \mathcal{I} \times \mathcal{V}$ if C_i is honest, and $(u_i, v_i) \in (\mathcal{I} \times \mathcal{V}) \cup \{(\perp, \perp)\}$ if C_i is corrupted:

1. Store $M := M \mid\mid [\![(i, u_i, v_i)]\!]$, and send $(\mathsf{InputReceived}, \mathsf{C}_i)$ to $\mathcal{S}_{\mathsf{Hist}}$.
2. All further input from C_i is ignored.

Evaluation This method must be called after initialization is completed. On input (Eval) from S_j, $j \in \{1, 2\}$:

1. Ignore all future client inputs.
2. Send $(\mathsf{EvalReceived}, \mathsf{S}_j)$ to $\mathcal{S}_{\mathsf{Hist}}$.

Once both servers have sent (Eval):

1. If one of S_1 and S_2 is corrupted:
   (a) Compute the set of client indices $I := \{i \mid (i, \_, \_) \in M\}$.
   (b) Receive a subset of client indices $(\mathsf{Influence\text{-}1}, J \subseteq I)$ from $\mathcal{S}_{\mathsf{Hist}}$ and filter the contributions: $M' := [\![(u, v) \mid (i, u, v) \in M \wedge i \in J]\!]$.

   If both S_1 and S_2 are honest, define $M' := [\![(u, v) \mid (\_, u, v) \in M]\!]$. Let $N' := |M'|$ be the number of remaining client contributions.
2. Add dummy contributions
   (a) Frequency dummies
      i. Do $D_{IF}^{(j)}, \_ \leftarrow \mathsf{SampleFrequencyDummies}(\mathcal{D}, T, \lambda_3, t_3, \mathsf{plain}, \perp)$ for $j \in \{1, 2\}$.
      ii. If S_j is corrupted, receive $(\mathsf{Influence\text{-}2}, D_{IF}^{(j)})$ from $\mathcal{S}_{\mathsf{Hist}}$, where $D_{IF}^{(j)}$ contains $\mathbb{F}_q$ elements. Send $(\mathsf{Leakage\text{-}1}, |D_{IF}^{(3-j)}|)$ to $\mathcal{S}_{\mathsf{Hist}}$.
   (b) Let $M'_U := [\![u \mid (u, \_) \in M']\!]$ be the list of the indices in M'; and let M''_U be a random permutation of $M'_U \mid\mid D_{IF}^{(1)} \mid\mid D_{IF}^{(2)}$.
   (c) Duplication dummies
      i. $D_{ID} \leftarrow \mathsf{SampleDuplicateDummies}(M''_U, r, p, \mathsf{plain}, \perp)$.
      ii. If S_1 or S_2 is corrupted, send $(\mathsf{Leakage\text{-}2}, |D_{ID}|)$ to $\mathcal{S}_{\mathsf{Hist}}$.
   (d) Set $M''' := M' \mid\mid \left[\!\left[(u, 0) \mid u \in (D_{IF}^{(1)} \mid\mid D_{IF}^{(2)} \mid\mid D_{ID})\right]\!\right]$.
3. Define the following:
   (a) Let $U := \{u \mid (u, \_) \in M'''\}$ be the set of occurring bucket indices, and let $B := |U|$ be the number of buckets.
   (b) For $u \in U$, let $V_u := \{v \mid (u, v) \in M'''\}$ be the multiset of values for the bucket with index u, and let $V := \biguplus_{u \in U} V_u$ be the multiset of overall occurring values.
4. Compute the anonymous histogram of M''', i.e., the (sorted) list
$$\mathcal{H}^A := \left[\!\left[n_u := |\{(x, y) \in M''' \mid x = u\}| \;\middle|\; u \in U\right]\!\right]. \tag{13}$$
   If one of S_1 and S_2 is corrupted, send $(\mathsf{Leakage\text{-}3}, \mathcal{H}^A)$ to $\mathcal{S}_{\mathsf{Hist}}$.
5. Compute the (normal) histogram of M''' as list, sorted by the first component (which is unique):
$$\mathcal{H} := \left[\!\left[(u, v) \;\middle|\; u \in U \wedge v := \textstyle\sum_{v_i \in V_u} v_i\right]\!\right]. \tag{14}$$
6. Add dummy buckets:
   (a) For $j \in \{1, 2\}$, $D_B^{(j)} \leftarrow \mathsf{SampleDummyBuckets}(\Delta, \lambda_2, t_2, \mathsf{plain}, \perp)$.
   (b) If S_j is corrupted, send $(\mathsf{Leakage\text{-}4}, |D_B^{(3-j)}|)$ to $\mathcal{S}_{\mathsf{Hist}}$, and receive $(\mathsf{Influence\text{-}3}, D_B^{(j)})$ from $\mathcal{S}_{\mathsf{Hist}}$, where $D_B^{(j)}$ is a list containing at most $2t_2\Delta$ elements from $[0, \Delta]$.[a]
   (c) Define $\mathcal{H}' := \mathcal{H} \mid\mid \left[\!\left[(\perp, v) \mid v \in D_B^{(1)} \mid\mid D_B^{(2)}\right]\!\right]$ and $B' := |\mathcal{H}'|$.
7. Compute the private histogram

(a) Write $[\![(u_1, v_1), \ldots, (u_{B'}, v_{B'})]\!] := \text{Shuffle}(\mathcal{H}')$.

(b) Sample noise.

i. For $j \in \{1, 2\}$ and $i \in [B']$, sample $\zeta_i^{(j)} \leftarrow \mathsf{TDLap}(\lambda_1, t_1)$.

ii. If S_j is corrupted, receive $(\mathsf{Influence\text{-}4}, (\zeta_i^{(j)})_{i\in[B']} \in [-t_1, t_1]^{B'})$ from $\mathcal{S}_{\mathsf{Hist}}$.

iii. Set $\zeta_i := \zeta_i^{(1)} + \zeta_i^{(2)}$ for $i \in [B']$.

(c) Compute the private histogram as sorted list:[b][c]

$$\mathcal{H}_I^P := [\![(i, u_i, v_i + \zeta_i) \mid v_i + \zeta_i \geq \tau_{\mathcal{F}}]\!]. \tag{15}$$

(d) Compute the threshold leakage as sorted list:

$$V^\perp := [\![(i, v_i + \zeta_i) \mid v_i + \zeta_i < \tau_{\mathcal{F}}]\!]. \tag{16}$$

8. If one of S_1 and S_2 is corrupted:
(a) Send $(\mathsf{Leakage\text{-}5}, V^\perp, \mathcal{H}_I^P)$ to $\mathcal{S}_{\mathsf{Hist}}$.
(b) If $\mathcal{S}_{\mathsf{Hist}}$ responds with (abort), stop the execution.
9. Let $\mathcal{H}^P := [\![(u, v) \mid (\_, u, v) \in \mathcal{H}_I^P]\!]$ be a sorted list. Send $(\mathsf{Output}, \mathcal{H}^P)$ to the uncorrupted server(s).

[a] This could be optimized that arbitrary values from $\mathbb{F}_q$ are accepted. If these values make it over the threshold after the noise is added, it will be noticed as dummy bucket since there is no index associated. This is ok because it will only happen for maliciously chosen dummy buckets.

[b] We need to leak the i since we do not shuffle again after adding the noise, and $\mathcal{S}_{\mathsf{Hist}}$ knows the malicious noise corresponding to each i.

[c] Note that by the choice of $\tau_{\mathcal{F}}$, we never have the case that $u_i = \perp$.

Fig. 2: Ideal functionality realized by our protocol Π_{Hist}.

3.3 Privacy Proof

Theorem 4. *The view of $\mathcal{S}_{\mathsf{Hist}}$ when interacting with $\mathcal{F}_{\mathsf{Hist}}$ is (ε, δ)-differentially private for the same parameters as in Theorem 3 and threshold $\tau = \Delta + 2 \cdot t_1 + 1$.*

Proof. We assume that all clients except for C_{i^*} are corrupted, so that the adversary knows (or can even select) the inputs $x_i = (u_i, v_i)$ for each client C_i, $i \neq i^*$. Hence, we consider differential privacy for two neighboring datasets M_0, M_1 that agree on all x_i for $i \neq i^*$, and differ in arbitrary $x_{i^*}^{(0)}, x_{i^*}^{(1)} \in \mathbb{F}_q \times [0, \Delta]$. Moreover, the ideal functionality $\mathcal{F}_{\mathsf{Hist}}$ takes the role of the interactive mechanism and the simulator $\mathcal{S}_{\mathsf{Hist}}$ is the adversary.

In the following, we show how to translate the first two influence queries of $\mathcal{S}_{\mathsf{Hist}}$ into a transformation of the input dataset that is independent of the concrete value of x_{i^*}. 1. In Step 1b of the interaction, the adversary is allowed to filter the dataset by specifying an index set J. 2. Then in Step 2(a)ii, the adversary submits a set of frequency dummies. This has the effect of extending the dataset with points $(d, 0) \in \mathcal{I} \times \mathcal{V}$ for each $d \in D_{IF}^{(j)}$.

Since both actions are independent of the value x_{i^*}, they will be the same for any two neighboring datasets M_0, M_1 that differ in position i^*. If we apply these actions to both datasets, we obtain two new datasets $\widehat{M}_0$ and $\widehat{M}_1$. If $i^* \notin J$, then $\widehat{M}_0 = \widehat{M}_1$ and the adversary's view in the following interaction is identical in both cases. Hence, from now on, we assume that $i^* \in J$. In this case, $\widehat{M}_0$ and $\widehat{M}_1$ will be neighboring datasets of size $|J| + |D_{IF}^{(j)}|$.

In the following, we show how to obtain the remainder of the transcript of $\langle \mathcal{F}_{\mathsf{Hist}}(M), \mathcal{S}_{\mathsf{Hist}} \rangle$ by post-processing the output of $\mathcal{M}_{\mathsf{Hist}}(\widehat{M})$, which is (ε, δ)-differentially private by Theorem 3. Let $(|D_{IF}|, |D_{ID}|, \mathcal{H}^A, |D_B|, V^\perp, \mathcal{H}^P) \leftarrow \mathcal{M}_{\mathsf{Hist}}(\widehat{M})$.

As the first leakage, we can directly forward the number of frequency dummies $|D_{IF}|$ (Step 2(a)ii), the number of duplications $|D_{ID}|$ (Step 2(c)ii), and the anonymous histogram $\mathcal{H}^A$ (Step 4) to $\mathcal{S}_{\mathsf{Hist}}$. These values are already distributed correctly, because $\mathcal{F}_{\mathsf{Hist}}$ samples the frequency dummies in the same way as $\mathcal{M}_{\mathsf{Hist}}$. Also when sampling the duplication dummies, each element of the original dataset and each frequency dummy is duplicated independently with the same distribution. Hence, it does not make a difference, if we treat the frequency dummies chosen by $\mathcal{S}_{\mathsf{Hist}}$ as zero-valued elements of $\widehat{M}$ as described above, and D_{ID} (and thus also $|D_{ID}|$) will be distributed correctly. Moreover, the anonymous histogram is also computed in the same way based on the dataset and the dummies.

In the next step, the private (non-anonymous) histogram is computed. First, we leak the number of dummy buckets $|D_B|$ to $\mathcal{S}_{\mathsf{Hist}}$ (Step 6b). Then, we receive the dummy buckets D_B^{c} (Step 6b), where $\mathsf{c} \in \{1,2\}$ is the index of the corrupted server, and the noise values $\boldsymbol{\zeta}^{(\mathsf{c})}$ (Step 7(b)ii) that $\mathcal{S}_{\mathsf{Hist}}$ has chosen for the corrupted server.

We now use these to post-process the threshold leakage $V^{\perp}$ and the histogram $\mathcal{H}^P$ that we obtained from $\mathcal{M}_{\mathsf{Hist}}$. Note that $\mathcal{F}_{\mathsf{Hist}}$ shuffles the histogram with the dummy buckets in Step 7a. So we also create a shuffled list, but while the values from $\mathcal{H}^P$ and $V^{\perp}$ already include honestly generated noise, we need to add the same noise to the dummies buckets that were provided by $\mathcal{S}_{\mathsf{Hist}}$:

$$A := \text{Shuffle}\left(\mathcal{H}^P \mid\mid \llbracket(\perp, v) \mid v \in V^{\perp}\rrbracket \mid\mid \llbracket(\perp, v_j + \zeta_j) \mid v_j \in D_B^{\mathsf{c}}, \zeta_j \leftarrow \mathsf{TDLap}(\lambda_1, t_1)\rrbracket\right).$$

Now we need to add the noise values $\zeta_i^{(\mathsf{c})}$ provided by $\mathcal{S}_{\mathsf{Hist}}$ and then simulate the thresholding step. Note that the threshold $\tau_{\mathcal{F}}$ used by $\mathcal{F}_{\mathsf{Hist}}$ is higher than $\tau_{\mathcal{M}}$ used in $\mathcal{M}_{\mathsf{Hist}}$. The difference is t_1, which is also the maximum value of the $\zeta_i^{(\mathsf{c})}$ noise values chosen by $\mathcal{S}_{\mathsf{Hist}}$. Hence, by adding the noise from $\mathcal{S}_{\mathsf{Hist}}$, it can never be the case that any values from $V^{\perp}$ make it over the threshold $\tau_{\mathcal{F}}$. Moreover, all the dummy buckets in $D_B^{(\mathsf{c})}$ are constraint to be in $[0, \varDelta]$, so they do not make it over the threshold either. Write $A =: \llbracket(u_1, v_1), \ldots, (u_{B'}, u_{B'})\rrbracket$. We compute

$$\begin{aligned} \widetilde{\mathcal{H}_I^P} &:= \llbracket(i, u_i, v_i + \zeta_i^{(\mathsf{c})}) \mid i \in [B'] \wedge v_i + \zeta_i^{(\mathsf{c})} \geq \tau_{\mathcal{F}}\rrbracket \qquad \text{and} \\ \widetilde{V^{\perp}} &:= \llbracket(i, v_i + \zeta_i^{(\mathsf{c})}) \mid i \in [B'] \wedge v_i + \zeta_i^{(\mathsf{c})} < \tau_{\mathcal{F}}\rrbracket, \end{aligned}$$

and send these as final output to $\mathcal{S}_{\mathsf{Hist}}$ (Step 8a).

Because the noise is independently added to each bucket, and the buckets have been shuffled, the distribution matches what $\mathcal{S}_{\mathsf{Hist}}$ seeds in its interaction with $\mathcal{F}_{\mathsf{Hist}}$. Hence, we have shown that the view of $\mathcal{S}_{\mathsf{Hist}}$ is (ε, δ)-differentially private, because we can generate the same distribution by post-processing the output of an (ε, δ)-differentially private mechanism $\mathcal{M}_{\mathsf{Hist}}$. □

4 Our Protocol

4.1 Subprotocols

In our histogram protocol, we use two major subprotocols, $\varPi_{\mathsf{Shuffle}}$ and $\varPi_{\mathsf{OPRF}}$. These do not realize UC functionalities, but should be seen as subroutines to make the presentation more modular. In $\varPi_{\mathsf{Shuffle}}$ (formally stated in Figure 14, Appendix G) the servers take turns in randomly permuting and rerandomizing a list of HSM-CL/ElGamal ciphertext pairs while proving correctness in zero-knowledge with $\varPi^{\mathsf{ZK}}_{\mathsf{Shuffle}}$. Common output is a new list of ciphertext pairs such that neither party knows how they correspond to the input list. The second protocol $\varPi_{\mathsf{OPRF}}$ (Figure 3) is used to obliviously evaluate the Dodis-Yampolskiy PRF on a list of encrypted inputs u_i under a randomly sampled key k such that both parties learn the output $t_i = F(k, u_i) = g^{\frac{1}{k+u_i}}$. A similar protocol appeared in [Mia+20] who used Camenisch-Shoup [CS03] encryption instead of HSM-CL.

Protocol $\varPi_{\mathsf{OPRF}}$

Common input: Sequence of ciphertexts $(\mathsf{clct}\text{-}u_i)_{i\in[n]}$.

1. Sample the OPRF key: S_1 and S_2 send $(\mathsf{CL\text{-}Random})$ to $\mathcal{F}_{\mathsf{Enc}}$ and receive a ciphertext $\mathsf{clct}\text{-}k$.
2. Prepare the OPRF evaluation: For $i \in [n]$, S_2 does:
   (a) Sample $r_i \in_R \mathbb{F}_q^*$ and $s_i, \rho_i \in_R \mathbb{F}_q$.
   (b) Compute $\mathsf{clct}\text{-}a_i := (\mathsf{clct}\text{-}u_i + \mathsf{clct}\text{-}k) \cdot_R^{s_i} r_i$ and $\mathsf{cm}_i \leftarrow \mathsf{PedCommit}(r_i; \rho_i)$.

(c) Compute a proof of correctness according to $\mathsf{R}_{\mathsf{AddBlindCom}}$:

$$\pi_i^{\mathsf{oprf1}} \leftarrow \mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{AddBlindCom}}.\mathsf{Prove}\big((\mathsf{clct}\text{-}k, \mathsf{clct}\text{-}u_i, \mathsf{clct}\text{-}a_i, \mathsf{cm}_i), (r_i, s_i, \rho_i)\big).$$

(d) Send $(\mathsf{clct}\text{-}a_i, \mathsf{cm}_i, \pi_i^{\mathsf{oprf1}})$ to S_1.
S_1 aborts if $\mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{AddBlindCom}}.\mathsf{Verify}\big(\pi_i^{\mathsf{oprf1}}, (\mathsf{clct}\text{-}k, \mathsf{clct}\text{-}u_i, \mathsf{clct}\text{-}a_i, \mathsf{cm}_i)\big) = \perp$ for any $i \in [n]$.

3. Perform the exponentiations: S_1 and S_2 send $(\mathsf{DecExpInv}, (\mathsf{clct}\text{-}a_1, \ldots, \mathsf{clct}\text{-}a_n))$ to $\mathcal{F}_{\mathsf{Enc}}$ such that both obtain $h_i = g^{\frac{1}{a_i}} \in \mathbb{G}$ for $i \in [n]$.
4. Complete the OPRF evaluation:
For $i \in [n]$, S_2 does:
(a) Compute $t_i := h_i^{r_i} \ \big(= g^{\frac{r_i}{a_i}} = g^{\frac{r_i}{r_i \cdot (u_i+k)}} = g^{\frac{1}{u_i+k}}\big)$.
(b) Compute a proof of correctness according to $\mathsf{R}_{\mathsf{ExpCom}}$:

$$\pi_i^{\mathsf{oprf2}} \leftarrow \mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{ExpCom}}.\mathsf{Prove}\big((h_i, t_i, \mathsf{cm}_i), (r_i, \rho_i)\big).$$

(c) Send $(t_i, \pi_i^{\mathsf{oprf2}})$ to S_1.
S_1 aborts if $\mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{ExpCom}}.\mathsf{Verify}\big(\pi_i^{\mathsf{oprf2}}, (h_i, t_i, \mathsf{cm}_i)\big) = \perp$ for any $i \in [n]$.

Common output: Sequence of PRF outputs $(t_i)_{i \in [n]}$.

Fig. 3: OPRF protocol in the $\mathcal{F}_{\mathsf{Enc}}$-hybrid model.

4.2 Computing Histograms

Our starting point is the protocol by Bell et al. [Bel+22] which is secure against semi-honest adversaries. We follow their blueprint and modify the protocol as needed to achieve malicious security. On a high level, the protocol $\mathit{\Pi}_{\mathsf{Hist}}$, which we formally specify in Figure 4, proceeds as follows: In an initialization phase, the servers jointly generate public keys for homomorphic encryption schemes such that the public keys gets published and each server obtains a share of the secret keys. Then, every client sends a single message with their encrypted index-value pair to both servers. Once the servers decide to compute a histogram of all collected client contributions, they start a two-party computation protocol: First, each server generates encrypted dummy contributions that have value zero such that they do not affect the final histogram. This is followed by a shuffle to mix the set real and dummy contributions. Then the servers run the OPRF protocol $\mathit{\Pi}_{\mathsf{OPRF}}$ to map each encrypted index to a random-looking tag. These tags are used to aggregate the values that are associated with the same index (without revealing either) into buckets, i.e., pairs of encrypted index and corresponding sums. Both servers add dummy buckets followed by another shuffle phase to mix real and dummy buckets. Finally the servers decrypt each sum and, if they are above a certain threshold, the corresponding index to obtain the histogram.

Why Proving Knowledge of Dummy Contributions Intuitively, if a malicious server produces invalid dummy contributions and buckets, that should not matter, because they only affect the other (honest) server's view, but not the output of the computation, and the simulator does not need to simulate the view of honest parties. We do not care that the malicious server's dummies were sampled according to the correct distribution, but we need to make sure that the server *knows* them. Otherwise, the server could just duplicate ciphertexts provided by an honest client. and obtain leakage that is biased depending on the client's data in a way that it is no longer differentially private. Moreover, for the security proof we need to extract the corrupted server's dummy indices and bucket values to pass them to the ideal functionality $\mathcal{F}_{\mathsf{Enc}}$.

Protocol $\mathit{\Pi}_{\mathsf{Hist}}$ realizing $\mathcal{F}_{\mathsf{Hist}}$

Init Each server S_j sends (KeyGen) to $\mathcal{F}_{\mathsf{Enc}}$. All parties receive $(\mathsf{clpk}, \mathsf{popk})$.
Client Input Every client C_i, $i \in N$, with input $(u_i, v_i) \in \mathcal{I} \times \mathcal{V} \subseteq \mathbb{F}_q^2$:

1. Encrypt inputs: $\mathsf{clct}\text{-}u_i \leftarrow \mathsf{Enc}^{\mathsf{cl}}(\mathsf{clpk}, u_i; r_i)$, $\mathsf{poct}\text{-}v_i \leftarrow \mathsf{Enc}^{\mathsf{po}}(\mathsf{popk}, v_i; s_i)$.
2. Prove correctness: $\pi_{C,i} \leftarrow \mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{Client}}.\mathsf{Prove}\big((\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i), (u_i, r_i, v_i, s_i)\big)$.
3. Send $(\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i, \pi_{C,i})$ to S_1 and S_2.

The servers ignore any additional messages from C_i.
Eval 1. Agree on client inputs – Each S_j does the following:
   (a) If C_i did not send a message, set $(\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i, \pi_{C,i}) = (\bot, \bot, \bot)$.
   (b) Send $[\![(\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i)]\!]_{i\in[N]}$ to the other server.
   (c) Let $[\![(\overline{\mathsf{clct}\text{-}u_i}, \overline{\mathsf{poct}\text{-}v_i})]\!]_{i\in[N]}$ denote the ciphertexts received from S_{3-j}. Define

$$J := \Big\{ i \in [N] \;\Big|\; \mathsf{clct}\text{-}u_i = \overline{\mathsf{clct}\text{-}u_i} \wedge \mathsf{poct}\text{-}v_i = \overline{\mathsf{poct}\text{-}v_i} \wedge \mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{Client}}.\mathsf{Verify}\big(\pi_{C,i}, (\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i)\big) = \top \Big\}. \quad (17)$$

   (d) Filter the valid contributions: Set $M' := [\![(\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}c_i)]\!]_{i\in J}$ and $N' := |J|$.

2. Add dummy contributions:
   (a) Both S_j send $(\mathsf{PO\text{-}Zero})$ to $\mathcal{F}_{\mathsf{Enc}}$ and receive a ciphertext $\mathsf{poct}\text{-}0$.
   (b) Frequency dummies: Each S_j does
      i. Compute

$$D^{(j)}_{IF}, \mathcal{W} \leftarrow \mathsf{SampleFrequencyDummies}(\mathcal{D}, T, \lambda_3, t_3, \mathsf{enc}, \mathsf{clpk})$$

      ii. Write $D^{(j)}_{IF} = [\![\mathsf{clct}\text{-}d^{(j)}_{F,1}, \ldots, \mathsf{clct}\text{-}d^{(j)}_{F,n}]\!]$ and $\mathcal{W} = [\![(d^{(j)}_{F,1}, r^{(j)}_{F,1}), \ldots, (d^{(j)}_{F,n}, r^{(j)}_{F,n})]\!]$ with $n := |D^{(j)}_{IF}|$, and prove knowledge of the plaintexts:

$$P^{(j)}_{IF} := \Big[\!\!\Big[\pi^{(j)}_{F,i} \leftarrow \mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{CL\text{-}PoPK}}.\mathsf{Prove}\big(\mathsf{clct}\text{-}d^{(j)}_{F,i}, (d^{(j)}_{F,i}, r^{(j)}_{F,i})\big) \mid i \in [n] \Big]\!\!\Big].$$

      iii. Let $\mathsf{id}_1, \mathsf{id}_2$ be fresh IDs. Send $(\mathsf{Commit}, \mathsf{id}_j, D^{(j)}_{IF}, P^{(j)}_{IF})$ to $\mathcal{F}_{\mathsf{Com}}$. Upon receipt of $(\mathsf{Committed}, \mathsf{id}_{3-j})$ from $\mathcal{F}_{\mathsf{Com}}$, send $(\mathsf{Open}, \mathsf{id}_j)$ to $\mathcal{F}_{\mathsf{Com}}$. Receive $(\mathsf{Opened}, \mathsf{id}_{3-j}, D^{(3-j)}_{IF}, P^{(3-j)}_{IF})$ from $\mathcal{F}_{\mathsf{Com}}$.
      iv. Verify the received $D^{(3-j)}_{IF}$ and proofs: Abort if for any $i \in [|D^{(3-j)}_{IF}|]$

$$\mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{CL\text{-}PoPK}}.\mathsf{Verify}\big(\pi^{(3-j)}_{F,i}, \mathsf{clct}\text{-}d^{(3-j)}_{F,i}\big) = \bot.$$

   (c) Shuffle client and frequency dummies inputs: Let $N'' := N' + |D^{(1)}_{IF}| + |D^{(2)}_{IF}|$ and

$$M'' \leftarrow \mathit{\Pi}_{\mathsf{Shuffle}}\Big(M' \,||\, [\![(\mathsf{clct}\text{-}d, \mathsf{poct}\text{-}0) \mid \mathsf{clct}\text{-}d \in D^{(1)}_{IF}]\!] \,||\, [\![(\mathsf{clct}\text{-}d, \mathsf{poct}\text{-}0) \mid \mathsf{clct}\text{-}d \in D^{(2)}_{IF}]\!] \Big).$$

   (d) Duplication dummies:
      i. Send (Rand, N'', p) to $\mathcal{F}_{\mathsf{Rand}}$ and receive $\mathsf{rnd}_{ID} \in \{0,1\}^{N''}$.
      ii. Locally compute $D_{ID} \leftarrow \mathsf{SampleDuplicateDummies}(M'', p; \mathsf{rnd}_{ID})$.
   (e) Shuffle the duplications into the ciphertexts: Let $N''' := N'' + |D_{ID}|$ and

$$[\![(\mathsf{clct}\text{-}u'_i, \mathsf{poct}\text{-}v'_i)]\!]_{i\in[N''']} \leftarrow \mathit{\Pi}_{\mathsf{Shuffle}}\Big(M'' \,||\, [\![(\mathsf{clct}\text{-}d, \mathsf{poct}\text{-}0) \mid \mathsf{clct}\text{-}d \in D_{ID}]\!] \Big).$$

3. Evaluate the PRF on the encrypted indices: Run $(t_1, \ldots, t_{i\in[N''']}) \leftarrow \mathit{\Pi}_{\mathsf{OPRF}}((\mathsf{clct}\text{-}u'_i)_{i\in[N''']})$.
4. Aggregate – Both parties locally compute the following:

(a) Define the set $T := \{t_i \mid i \in [N''']\} \subseteq \mathbb{G}$, and for each $t \in T$ set $I_t := \{i \in [N'''] \mid t_i = t\}$ and $i_t := \min(I_t)$.
(b) Compute $\mathsf{clct}\text{-}u_t := \mathsf{clct}\text{-}u'_{i_t}$ and $\mathsf{poct}\text{-}v_t := \sum_{i \in I_t} \mathsf{poct}\text{-}v'_i$ for $t \in T$.
(c) Set $H := [\![(\mathsf{clct}\text{-}u_t, \mathsf{poct}\text{-}v_t)]\!]_{t \in T}$ and $B := |H|$ (number of buckets).

5. Add dummy buckets
   (a) Let $u_D \in \mathcal{D}$ be a public constant. Both S_j send $(\mathsf{CL\text{-}Constant}, u_D)$ to $\mathcal{F}_{\mathsf{Enc}}$ and receive a ciphertext $\mathsf{clct}\text{-}u_D$.
   (b) Each S_j computes

$$D_B^{(j)}, \mathcal{W} \leftarrow \mathsf{SampleDummyBuckets}(\Delta, \lambda_2, t_2, \mathsf{enc}, \mathsf{popk})$$

.
   (c) Write $D_B^{(j)} = [\![\mathsf{poct}\text{-}d_{B,1}^{(j)}, \ldots, \mathsf{poct}\text{-}d_{B,n}^{(j)}]\!]$ and $\mathcal{W} = [\![(d_{B,1}^{(j)}, r_{B,1}^{(j)}), \ldots, (d_{B,n}^{(j)}, r_{B,n}^{(j)})]\!]$ with $n := |D_B^{(j)}|$, and prove knowledge of the plaintexts:

$$P_B^{(j)} := \left[\!\!\left[\pi_{B,i}^{(j)} \leftarrow \Pi^{\mathsf{ZK}}_{\mathsf{Range}\text{-}[0,\Delta]}.\mathsf{Prove}\big(\mathsf{poct}\text{-}d_{B,i}^{(j)}, (d_{B,i}^{(j)}, r_{B,i}^{(j)})\big) \mid i \in [n]\right]\!\!\right].$$

   and sends $D_B^{(j)}, P_B^{(j)}$ to S_j.
   (d) Verify the received $D_B^{(3-j)}$ and proofs: Abort if for any $i \in [|D_B^{(3-j)}|]$

$$\Pi^{\mathsf{ZK}}_{\mathsf{Range}\text{-}[0,\Delta]}.\mathsf{Verify}\big(\pi_{B,i}^{(3-j)}, \mathsf{poct}\text{-}d_{B,i}^{(3-j)}\big) = \perp.$$

   (e) Set $H' := H \mid\mid [\![(\mathsf{clct}\text{-}u_D, \mathsf{poct}\text{-}d) \mid \mathsf{poct}\text{-}d \in D_B^{(1)}]\!] \mid\mid [\![(\mathsf{clct}\text{-}u_D, \mathsf{poct}\text{-}d) \mid \mathsf{poct}\text{-}d \in D_B^{(2)}]\!]$ with $B' := |H'|$.
6. Shuffle the buckets: Run $[\![(\mathsf{clct}\text{-}\bar{u}_i, \mathsf{poct}\text{-}\bar{v}_i)]\!]_{i \in [B']} \leftarrow \Pi_{\mathsf{Shuffle}}(H')$.
7. Add noise to the output – Every S_j does:
   (a) Sample $\zeta_i^{(j)} \leftarrow \mathsf{TDLap}(\lambda_1, t_1)$ for $i \in [B']$.
   (b) Encrypt $\mathsf{poct}\text{-}\zeta_i^{(j)} \leftarrow \mathsf{Enc}^{\mathsf{PO}}(\mathsf{popk}, \zeta_i^{(j)}; r_{\zeta,i}^{(j)})$, prove

$$\pi_{\zeta,i} \leftarrow \Pi^{\mathsf{ZK}}_{\mathsf{Range}\text{-}[-t_1,t_1]}.\mathsf{Prove}\big(\mathsf{poct}\text{-}\zeta_i^{(j)}, (\zeta_i^{(j)}, r_{\zeta,i}^{(j)})\big),$$

   and send $\mathsf{poct}\text{-}\zeta_i^{(j)}, \pi_{\zeta,i}$ to the other server for $i \in [B']$.
   (c) Verify the received ciphertexts and proofs: Abort if for any $i \in [B']$

$$\Pi^{\mathsf{ZK}}_{\mathsf{Range}\text{-}[-t_1,t_1]}.\mathsf{Verify}\big(\pi_{\zeta,i}^{(3-j)}, \mathsf{poct}\text{-}\zeta_i^{(3-j)}\big) = \perp.$$

   (d) Locally compute $\mathsf{poct}\text{-}\tilde{v}_i := \mathsf{poct}\text{-}\bar{v}_i + \mathsf{poct}\text{-}\zeta_i^{(1)} + \mathsf{poct}\text{-}\zeta_i^{(2)}$.
8. Threshold – Both servers:
   (a) Send $(\mathsf{PO\text{-}Decrypt}, \mathsf{poct}\text{-}\tilde{v}_i)$ to $\mathcal{F}_{\mathsf{Enc}}$ and receive $\tilde{v}_i \in \mathbb{F}_q$ for $i \in [B']$.[a]
   (b) Set $I := \{i \in [B'] \mid \tilde{v}_i \geq \tau_{\mathcal{F}}\}$ where $\tau_{\mathcal{F}} = \Delta + 2 \cdot t_1 + 1$.
   (c) Send $(\mathsf{CL\text{-}Decrypt}, \mathsf{clct}\text{-}\bar{u}_i)$ to $\mathcal{F}_{\mathsf{Enc}}$ and receive $\bar{u}_i$ for $i \in I$.
9. Compute the histogram: Both S_j compute and output the histogram (as sorted list):

$$\mathcal{H}^P := [\![(\tilde{u}_i, \tilde{v}_i) \mid i \in I]\!]. \tag{18}$$

[a] Decryption failures do not happen due to the proofs that the encrypted values are small enough.

Fig. 4: Histogram Protocol in the $\mathcal{F}_{\mathsf{Enc}}$-hybrid model.

4.3 Proof of Security

Theorem 5. *Given that we have zero-knowledge proofs for all relations etc.*

The protocol Π_{Hist} (Figure 4) securely UC-realizes the functionality $\mathcal{F}_{\mathsf{Hist}}$ (Figure 2) in the $(\mathcal{F}_{\mathsf{Enc}}, \ldots)$-hybrid model with computational security, tolerating active and static corruptions of any number of clients and at most one server.

Our proof strategy is similar to the security proof of SPDZ [Dam+12] and the MPC protocol in [BDO23].

Since we are in the $\mathcal{F}_{\mathsf{Enc}}$-hybrid model, the simulator $\mathcal{S}_{\mathsf{Hist}}$ simulates an instance of this functionality and can generate the encryption keys such that it knows the secret keys. Hence, it can extract the corrupted parties' inputs by decrypting them. For all ciphertexts generated by honest parties, $\mathcal{S}_{\mathsf{Hist}}$ uses encryptions of zero. Once it obtains the necessary information from $\mathcal{F}_{\mathsf{Hist}}$ (e.g., the final output or intermediate leakage) it can simulate the decryption operation by letting $\mathcal{F}_{\mathsf{Enc}}$ return the appropriate values.

To argue that this simulation is indeed indistinguishable from the real protocol, we need to use the security of the encryption scheme and the non-adaptive security of the Dodis-Yampolskiy PRF.

For the former, we use that the encryption scheme admits so-called "lossy" public keys. These are indistinguishable from real public keys, but encrypting and value results in an encryption of 0, i.e., the ciphertexts do not contain any information about the encrypted value. In the reduction, we use an environment $\mathcal{Z}$ that can distinguish the simulation from the real protocol to build a distinguisher between lossy and normal public keys.

Proof. Due to space reasons, the formal description of the simulator $\mathcal{S}_{\mathsf{Hist}}$ is given in Figures 15, 16, and 17 in Appendix H, and only provide a summary here: we need to prove that the simulation is indeed indistinguishable from the real execution, i.e., no computationally bounded environment $\mathcal{Z}$ can tell them apart. Let $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{real}}(\lambda)$ denote $\mathcal{Z}$'s interaction with the real protocol Π_{Hist} and its output is whatever $\mathcal{Z}$ outputs. Likewise, define $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{ideal}}(\lambda)$ as $\mathcal{Z}$'s interaction with the simulator $\mathcal{S}_{\mathsf{Hist}}$. We need to show that

$$\mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z}}(\lambda) := \left|\Pr\left[\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{real}}(\lambda) = 1\right] - \Pr\left[\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{ideal}}(\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

We want to prove indistinguishability of the simulation in two steps, where we use the security of the encryption schemes and of the PRF, respectively. These are intertwined however, since the protocol uses an encryption of the PRF key. Hence, we define two hybrid execution, $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid\text{-}1}}(\lambda)$ and $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid\text{-}2}}(\lambda)$, and prove indistinguishability in three steps using the security of the encryption schemes twice.

Simulation Overview $\mathcal{S}_{\mathsf{Hist}}$ simulates ciphertexts sent by honest parties by encryptions of zero. Since $\mathcal{S}_{\mathsf{Hist}}$ simulates an instance of $\mathcal{F}_{\mathsf{Enc}}$, it also generates the key pairs and, therefore, can use the secret keys $\mathsf{clsk}, \mathsf{posk}$ to extract the inputs of corrupted clients as well as dummies generated by the corrupted server. For most of the simulation, $\mathcal{S}_{\mathsf{Hist}}$ lets the simulated honest parties follow the instructions in Π_{Hist}. When it comes to the OPRF evaluations in Π_{OPRF} it needs to produce outputs that match the input submitted by the clients. Here it used the anonymous histogram $\mathcal{H}^A$ leaked by $\mathcal{F}_{\mathsf{Hist}}$, to simulate random PRF outputs accordingly, i.e., if there are l indices in the input that appear k times each, then $\mathcal{S}_{\mathsf{Hist}}$ will simulate that l random group elements appear k times each in the output of Π_{Hist}. Finally, when decrypting the aggregated values and indices (if the values make it over the threshold), $\mathcal{S}_{\mathsf{Hist}}$ uses the leaked $V^{\perp}$ and $\mathcal{H}^P_I$ obtained from $\mathcal{F}_{\mathsf{Enc}}$ to simulate $\mathcal{F}_{\mathsf{Enc}}$ returning the corresponding values.

Hybrid-1 The first hybrid $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid\text{-}1}}(\lambda)$, identical to $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{real}}(\lambda)$ except for the following modification: In the real protocol, the parties use an encryption of the PRF key $\mathsf{clct}\text{-}k$ in the OPRF subprotocol. Now we patch the execution such that the F^{DY} is evaluated using an unrelated key: We use clsk generated by $\mathcal{F}_{\mathsf{Enc}}$ to decrypt the inputs $u_i \leftarrow \mathsf{Dec}^{\mathsf{cl}}(\mathsf{clsk}, \mathsf{clct}\text{-}u_i)$ for $i \in [n]$ of the OPRF subprotocol Π_{OPRF} and sample a fresh PRF key $k' \in_R \mathbb{F}_q$. If S_1 is corrupted, we let S_2 in Step 4c of Π_{OPRF}) send $t'_i := \mathsf{F}^{\mathsf{DY}}(k', u_i)$ instead of $t_i = h_i^{r_i}$ and simulate the corresponding proof. This is indistinguishable, since the r_i are uniformly random in $\mathbb{F}_q^*$. If S_2 is corrupted, we extract $r_i = a_i \cdot (u_i + k)^{-1}$ and then let $\mathcal{F}_{\mathsf{Enc}}$ in Step 3 output $h'_i := (\mathsf{F}^{\mathsf{DY}}(k', u_i))^{1/r_i}$ such that again $t'_i = \mathsf{F}^{\mathsf{DY}}(k', u_i)$. In both cases, the output of the modified Π_{OPRF} is now the output of F^{DY} under a fresh random key $k' \in_R \mathbb{F}_q$.

This is indistinguishable due to the security of the encryption scheme and because the parties only ever see an encryption of the PRF key k; it is never revealed in plain. Moreover, while S_1 sees values related to the key, they are randomized with the masks r_i that are only known to S_2.

Claim 1. *The environment's view in the real execution is computationally indistinguishable from the view in the first hybrid execution:*

$$\mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},1}(\lambda) := \left|\Pr\left[\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{real}}(\lambda) = 1\right] - \Pr\left[\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid\text{-}1}}(\lambda) = 1\right]\right| \le \mathsf{negl}(\lambda).$$

Hybrid-2 The second hybrid $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid\text{-}2}}(\lambda)$ is identical to the first, except that we replace the Dodis-Yampolskiy PRF with a random function. We first (lazily) sample a random function $H\colon \mathbb{F}_q \to \mathbb{G}$. Then we modify the execution in the same way as above, but injecting outputs of the random function $H(u_i)$ instead of $\mathsf{F}^{\mathsf{DY}}(k', u_i)$ such that the output of the modified $\mathit{\Pi}_{\mathsf{OPRF}}$ is now the output of H.

We argue indistinguishability by using the non-adaptive security of the Dodis-Yampolskiy PRF. Non-adaptive security is sufficient, since the inputs $(u_i)_{i\in[n]}$ are fixed before the key k' is sampled.

Claim 2. *Given that the Dodis-Yampolskiy PRF is an N'''-non-adaptively secure PRF, the environment's views in the two hybrid executions are computationally indistinguishable:*

$$\mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},2}(\lambda) := \left|\Pr\left[\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid\text{-}1}}(\lambda) = 1\right] - \Pr\left[\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid\text{-}2}}(\lambda) = 1\right]\right| \le \mathsf{negl}(\lambda).$$

UC Simulation Finally, we show that the second hybrid is indistinguishable to our simulation with the simulator $\mathcal{S}_{\mathsf{Hist}}$ given in Figure 15. We show indistinguishability based on the security of the encryption schemes:

Claim 3. *The environment's view in the second hybrid execution is computationally indistinguishable from the view in the simulation:*

$$\mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},3}(\lambda) := \left|\Pr\left[\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid\text{-}2}}(\lambda) = 1\right] - \Pr\left[\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{ideal}}(\lambda) = 1\right]\right| \le \mathsf{negl}(\lambda).$$

Due to space reasons, the formal proofs of the three claims are given in Appendix I. Combining the three claims, we use the triangle inequality to bound

$$\mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z}}(\lambda) \le \mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},1}(\lambda) + \mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},2}(\lambda) + \mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},3}(\lambda) \le \mathsf{negl}(\lambda),$$

which concludes the proof of Theorem 5. □

5 Evaluation

In this section, we compute the concrete communication overhead of our protocol for various values of ε and numbers of client inputs N. For the sizes of CL ciphertexts and group elements, we used BICYCL [Bou+23, Tables 2 and 3]. We use the Bulletproofs-based construction from Acorn [Bel+23, Section 5.3.2] for range proofs, and Bayer-Groth [BG12, Table 1] for shuffle proofs, plugging in the appropriate group and scalar sizes for our protocol.

Our results are shown in Table 1. As expected, communication grows quadratically with $1/\varepsilon$ for small N. However, as the total number of dummies remains sub-linear in N, this impact becomes less severe as N grows.

Table 1: Communication cost per client (in bytes) for each server of our $\mathit{\Pi}_{\mathsf{Hist}}$ for different values of N, ε, and $\delta = 10^{-9}, \lambda = 128$, and $\sigma = 40$.

ε	10^5	10^6	10^7	10^8	10^9
4	403476	45205	9106	5114	4554
2	8744547	878653	92063	13404	5538
1	217639887	21768184	2181013	222296	26424
0.5	5956609178	595665111	59570705	5961264	600320

Acknowledgements Part of the work was done while Lennart Braun worked as research intern at Google. Afterwards his research was supported by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC), and the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0085. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA). Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

References

[Add+22] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. "Prio+: Privacy Preserving Aggregate Statistics via Boolean Shares". In: *SCN*. Vol. 13409. Lecture Notes in Computer Science. Springer, 2022, pp. 516–539.

[AG21] Apple and Google. *Exposure Notification Privacy-preserving Analytics (ENPA)*. 2021. URL: https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf.

[Bar+23] Richard Barnes, David Cook, Christopher Patton, and Phillipp Schoppmann. *Verifiable Distributed Aggregation Functions*. Internet-Draft draft-irtf-cfrg-vdaf-08. Work in Progress. Internet Engineering Task Force, Nov. 2023. 114 pp. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-vdaf/08/.

[Bas+20] Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Thakurta. "Practical Locally Private Heavy Hitters". In: *J. Mach. Learn. Res.* 21 (2020), 16:1–16:42.

[BDO23] Lennart Braun, Ivan Damgård, and Claudio Orlandi. "Secure Multiparty Computation from Threshold Encryption Based on Class Groups". In: *CRYPTO 2023, Part I*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14081. LNCS. Springer, Heidelberg, Aug. 2023, pp. 613–645. DOI: 10.1007/978-3-031-38557-5_20.

[Bel+20] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. "Secure Single-Server Aggregation with (Poly)Logarithmic Overhead". In: *ACM CCS 2020*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. ACM Press, Nov. 2020, pp. 1253–1269. DOI: 10.1145/3372297.3417885.

[Bel+22] James Bell, Adrià Gascón, Badih Ghazi, Ravi Kumar, Pasin Manurangsi, Mariana Raykova, and Phillipp Schoppmann. "Distributed, Private, Sparse Histograms in the Two-Server Model". In: *ACM CCS 2022*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. ACM Press, Nov. 2022, pp. 307–321. DOI: 10.1145/3548606.3559383.

[Bel+23] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. "ACORN: Input Validation for Secure Aggregation". In: *USENIX Security Symposium*. USENIX Association, 2023, pp. 4805–4822.

[BG12] Stephanie Bayer and Jens Groth. "Efficient Zero-Knowledge Argument for Correctness of a Shuffle". In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 263–280. DOI: 10.1007/978-3-642-29011-4_17.

[BH12] Itay Berman and Iftach Haitner. "From Non-adaptive to Adaptive Pseudorandom Functions". In: *TCC 2012*. Ed. by Ronald Cramer. Vol. 7194. LNCS. Springer, Heidelberg, Mar. 2012, pp. 357–368. DOI: 10.1007/978-3-642-28914-9_20.

[BNS19] Mark Bun, Kobbi Nissim, and Uri Stemmer. "Simultaneous Private Learning of Multiple Concepts". In: *JMLR* 20 (2019), 94:1–94:34.

[Bon+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 1175–1191. DOI: 10.1145/3133956.3133982.

[Bon+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. "Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs". In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 67–97. DOI: 10.1007/978-3-030-26954-8_3.

[Bon+21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. "Lightweight Techniques for Private Heavy Hitters". In: *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2021, pp. 762–776. DOI: 10.1109/SP40001.2021.00048.

[Bou+23] Cyril Bouvier, Guilhem Castagnos, Laurent Imbert, and Fabien Laguillaumie. "I Want to Ride My BICYCL : BICYCL Implements CryptographY in CLass Groups". In: *J. Cryptol.* 36.3 (2023), p. 17.

[BS15] Raef Bassily and Adam D. Smith. "Local, Private, Efficient Protocols for Succinct Histograms". In: *47th ACM STOC*. Ed. by Rocco A. Servedio and Ronitt Rubinfeld. ACM Press, June 2015, pp. 127–135. DOI: 10.1145/2746539.2746632.

[Bün+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[CB17] Henry Corrigan-Gibbs and Dan Boneh. "Prio: Private, Robust, and Scalable Computation of Aggregate Statistics". In: *NSDI*. USENIX Association, 2017, pp. 259–282.

[CL15] Guilhem Castagnos and Fabien Laguillaumie. "Linearly Homomorphic Encryption from DDH". In: *CT-RSA 2015*. Ed. by Kaisa Nyberg. Vol. 9048. LNCS. Springer, Heidelberg, Apr. 2015, pp. 487–505. DOI: 10.1007/978-3-319-16715-2_26.

[CLT18] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. "Practical Fully Secure Unrestricted Inner Product Functional Encryption Modulo p". In: *ASIACRYPT 2018, Part II*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11273. LNCS. Springer, Heidelberg, Dec. 2018, pp. 733–764. DOI: 10.1007/978-3-030-03329-3_25.

[Cra97] Ronald Cramer. "Modular Design of Secure yet Practical Cryptographic Protocols". PhD thesis. Universiteit van Amsterdam, Jan. 31, 1997.

[CS03] Jan Camenisch and Victor Shoup. "Practical Verifiable Encryption and Decryption of Discrete Logarithms". In: *CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. LNCS. Springer, Heidelberg, Aug. 2003, pp. 126–144. DOI: 10.1007/978-3-540-45146-4_8.

[Dam+12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38.

[Dav+23] Hannah Davis, Christopher Patton, Mike Rosulek, and Phillipp Schoppmann. "Verifiable Distributed Aggregation Functions". In: *Proc. Priv. Enhancing Technol.* 2023.4 (2023), pp. 578–592.

[DR14] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Foundations and Trends in Theoretical Computer Science* 9.3-4 (2014), pp. 211–407. DOI: 10.1561/0400000042.

[Dwo+06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. "Calibrating Noise to Sensitivity in Private Data Analysis". In: *TCC 2006*. Ed. by Shai Halevi and Tal Rabin. Vol. 3876. LNCS. Springer, Heidelberg, Mar. 2006, pp. 265–284. DOI: 10.1007/11681878_14.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. "A Verifiable Random Function with Short Proofs and Keys". In: *PKC 2005*. Ed. by Serge Vaudenay. Vol. 3386. LNCS. Springer, Heidelberg, Jan. 2005, pp. 416–431. DOI: 10.1007/978-3-540-30580-4_28.

[ElG84] Taher ElGamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *CRYPTO'84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. LNCS. Springer, Heidelberg, Aug. 1984, pp. 10–18.

[Fis05] Marc Fischlin. "Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors". In: *CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. LNCS. Springer, Heidelberg, Aug. 2005, pp. 152–168. DOI: 10.1007/11535218_10.

[FS87] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. LNCS. Springer, Heidelberg, Aug. 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12.

[Geo+23] Tim Geoghegan, Christopher Patton, Brandon Pitman, Eric Rescorla, and Christopher A. Wood. *Distributed Aggregation Protocol for Privacy Preserving Measurement*. Internet-Draft draft-ietf-ppm-dap-09. Work in Progress. Internet Engineering Task Force, Dec. 2023. 77 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-ppm-dap/09/.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. "How to construct random functions". In: *J. ACM* 33.4 (1986), pp. 792–807. DOI: 10.1145/6490.6503.

[Gha+21] 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: *EUROCRYPT*. 2021, pp. 463–488.

[Hem+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. "Adaptively Secure Garbled Circuits from One-Way Functions". In: *CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 149–178. DOI: 10.1007/978-3-662-53015-3_6.

[HKR19] Max Hoffmann, Michael Klooß, and Andy Rupp. “Efficient Zero-Knowledge Arguments in the Discrete Log Setting, Revisited”. In: *ACM CCS 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 2093–2110. DOI: 10.1145/3319535.3354251.

[Int22] Internet Security Research Group. *Divvi Up: A privacy-respecting system for aggregate statistics*. 2022. URL: https://divviup.org/.

[JL09] Stanislaw Jarecki and Xiaomin Liu. “Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection”. In: *TCC 2009*. Ed. by Omer Reingold. Vol. 5444. LNCS. Springer, Heidelberg, Mar. 2009, pp. 577–594. DOI: 10.1007/978-3-642-00457-5_34.

[Kor+09] Aleksandra Korolova, Krishnaram Kenthapadi, Nina Mishra, and Alexandros Ntoulas. “Releasing Search Queries and Clicks Privately”. In: *WWW*. 2009.

[Li+23] Hanjun Li, Huijia Lin, Antigoni Polychroniadou, and Stefano Tessaro. “LERNA: Secure Single-Server Aggregation via Key-Homomorphic Masking”. In: *ASIACRYPT (1)*. Vol. 14438. Lecture Notes in Computer Science. Springer, 2023, pp. 302–334.

[Ma+23] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. “Flamingo: Multi-Round Single-Server Secure Aggregation with Applications to Private Federated Learning”. In: *2023 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2023, pp. 477–496. DOI: 10.1109/SP46215.2023.10179434.

[Mia+20] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. “Two-Sided Malicious Security for Private Intersection-Sum with Cardinality”. In: *CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Heidelberg, Aug. 2020, pp. 3–33. DOI: 10.1007/978-3-030-56877-1_1.

[Moz22] Mozilla. *Origin Telemetry*. 2022. URL: https://firefox-source-docs.mozilla.org/toolkit/components/telemetry/collection/origin.html.

[MP04] Ueli M. Maurer and Krzysztof Pietrzak. “Composition of Random Systems: When Two Weak Make One Strong”. In: *TCC 2004*. Ed. by Moni Naor. Vol. 2951. LNCS. Springer, Heidelberg, Feb. 2004, pp. 410–427. DOI: 10.1007/978-3-540-24638-1_23.

[MST23] Dimitris Mouris, Pratik Sarkar, and Nektarios Georgios Tsoutsos. *PLASMA: Private, Lightweight Aggregated Statistics against Malicious Adversaries with Full Security*. Cryptology ePrint Archive, Report 2023/080. https://eprint.iacr.org/2023/080. 2023.

[Mye04] Steven Myers. “Black-Box Composition Does Not Imply Adaptive Security”. In: *EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. LNCS. Springer, Heidelberg, May 2004, pp. 189–206. DOI: 10.1007/978-3-540-24676-3_12.

[Rot+19] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. “Honeycrisp: large-scale differentially private aggregation without a trusted core”. In: *SOSP*. ACM, 2019, pp. 196–210.

[ST20] Pratik Soni and Stefano Tessaro. “On the Query Complexity of Constructing PRFs from Non-adaptive PRFs”. In: *SCN 20*. Ed. by Clemente Galdi and Vladimir Kolesnikov. Vol. 12238. LNCS. Springer, Heidelberg, Sept. 2020, pp. 546–565. DOI: 10.1007/978-3-030-57990-6_27.

[VZ23] Salid Vadhan and Wanrong Zhang. “Concurrent Composition Theorems for Differential Privacy”. In: *55th ACM STOC*. ACM Press, June 2023, pp. 507–519. DOI: 10.1145/3564246.3585241.

[Zhu+20] Wennan Zhu, Peter Kairouz, Brendan McMahan, Haicheng Sun, and Wei Li. “Federated Heavy Hitters Discovery with Differential Privacy”. In: *AISTATS*. Vol. 108. Proceedings of Machine Learning Research. PMLR, 2020, pp. 3837–3847.

[ZXX16] Jun Zhang, Xiaokui Xiao, and Xing Xie. “PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions”. In: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD ’16. 2016.

A Encryption Schemes

The encryption schemes are defined in Figures 5 and 6. The following theorems state the security properties that we need.

Theorem 6 (Security of modified HSM-CL [BDO23]). *Under the* HSM *assumption, the* $\mathit{\Pi}_{\mathsf{HSM\text{-}CL}}$ *encryption scheme*

1. *provides indistinguishability under chosen plaintext attacks (IND-CPA), and*
2. *has lossy public keys which are indistinguishable from real public keys.*

Theorem 7 (Security of modified ElGamal). *Under the* DDH *assumption, the* $\mathit{\Pi}_{\mathsf{ElGamal}}$ *encryption scheme*

1. *provides indistinguishability under chosen plaintext attacks (IND-CPA), and*
2. *has lossy public keys which are indistinguishable from real public keys.*

- $\mathsf{KeyGen}_b^{\mathsf{cl}}(\mathsf{pp}_{\mathsf{cl}})$ for $b \in \{0, 1\}$:
  1. Sample $\mathsf{clsk}, \beta \leftarrow \mathcal{D}_q$.
  2. Set $\overline{\mathsf{clpk}} := g_q^{\mathsf{clsk}}$
  3. Set $\mathsf{clpkct} := (g_q^{\beta}, f^b \cdot \overline{\mathsf{clpk}}^{\beta})$ and $\mathsf{clpk} := (\overline{\mathsf{clpk}}, \mathsf{clpkct})$.
  4. Output $(\mathsf{clsk}, \mathsf{clpk})$.
- $\mathsf{Enc}^{\mathsf{cl}}(\mathsf{clpk}, m \in \mathbb{F}_q)$:
  1. Parse $\mathsf{clpk} = (\overline{\mathsf{clpk}}, \mathsf{clpkct})$.
  2. Sample $r \leftarrow \mathcal{D}_q$.
  3. Output $\mathsf{clct} = (\mathsf{clpkct}_1^m \cdot g_q^r, \mathsf{clpkct}_2^m \cdot \overline{\mathsf{clpk}}^r)$.
- $\mathsf{Dec}^{\mathsf{cl}}(\mathsf{clsk}, \mathsf{clct})$:
  1. Compute $M := \mathsf{clct}_2 \cdot (\mathsf{clct}_1)^{-\mathsf{clsk}}$
  2. If $M \in F$, then output $m := \log_f(M)$, and otherwise output $m := \bot$.
- $\mathsf{PDec}^{\mathsf{cl}}(\mathsf{clsk}_i, \mathsf{clct})$:
  1. Compute $\mathsf{clct}_2' := \mathsf{clct}_2 \cdot (\mathsf{clct}_1)^{-\mathsf{clsk}_i}$.
  2. Output $\mathsf{clct}' := (\mathsf{clct}_1, \mathsf{clct}_2')$.
- $\mathsf{Randomize}^{\mathsf{cl}}(\mathsf{clpk}, \mathsf{clct}; r)$:
  1. Sample $r \leftarrow \mathcal{D}_q$.
  2. Output $\mathsf{clct}' := (\mathsf{clct}_1 \cdot g_q^r, \mathsf{clct}_2 \cdot \mathsf{clpk}^r)$.

Fig. 5: The HSM-CL encryption scheme [CLT18] modified to have normal/lossy public keys as in [BDO23].

```
– $\mathsf{KeyGen}^{\mathsf{po}}_b(\mathsf{pp}_{\mathsf{po}})$ for $b \in \{0, 1\}$:
    1. Sample $\mathsf{posk}, \beta \leftarrow \mathbb{F}_q$.
    2. Set $\overline{\mathsf{popk}} := g^{\mathsf{posk}}$
    3. Set $\mathsf{popkct} := (g^{\beta}, g^{b} \cdot \overline{\mathsf{popk}}^{\beta})$ and $\mathsf{popk} := (\overline{\mathsf{popk}}, \mathsf{popkct})$.
    4. Output $(\mathsf{posk}, \mathsf{popk})$.
– $\mathsf{Enc}^{\mathsf{po}}(\mathsf{popk}, m \in \mathbb{F}_q)$:
    1. Parse $\mathsf{popk} = (\overline{\mathsf{popk}}, \mathsf{popkct})$.
    2. Sample $r \leftarrow \mathbb{F}_q$.
    3. Output $\mathsf{poct} = (\mathsf{popkct}_1^m \cdot g^r, \mathsf{popkct}_2^m \cdot \overline{\mathsf{popk}}^r)$.
– $\mathsf{Dec}^{\mathsf{po}}(\mathsf{posk}, \mathsf{poct})$:
    1. compute $M := \mathsf{poct}_2 \cdot (\mathsf{poct}_1)^{-\mathsf{posk}}$
    2. run $m := \log_g(M)$ (for limited time)
    3. output $m \in \{x \in \mathbb{F}_q \mid x \text{ small}\} \cup \{\perp\}$
– $\mathsf{PDec}^{\mathsf{po}}(\mathsf{posk}_i, \mathsf{poct})$:
    1. compute $\mathsf{poct_2}' := \mathsf{poct}_2 \cdot (\mathsf{poct}_1)^{-\mathsf{posk}_i}$
    2. output $\mathsf{poct}' := (\mathsf{poct}_1, \mathsf{poct_2}')$
– $\mathsf{Randomize}^{\mathsf{po}}(\mathsf{popk}, \mathsf{poct}; r)$:
    1. sample $r \in \mathbb{F}_q$
    2. set $\mathsf{poct}'_1 := \mathsf{poct}_1 \cdot g^r$
    3. set $\mathsf{poct}'_2 := \mathsf{poct}_2 \cdot \mathsf{popk}^r$
    4. output $\mathsf{poct}' := (\mathsf{poct}'_1, \mathsf{poct}'_2)$
```

Fig. 6: The ElGamal encryption scheme [ElG84] modified to have normal/lossy public keys analogous to HSM-CL in Figure 5.

B Non-Adaptive Pseudorandom Functions

Recall that in the standard definition of a pseudorandom function (PRF) ([GGM86], see Definition 6) the adversary $\mathcal{A}$ is a PPT oracle machine. It must distinguish whether the oracle is the PRF $\mathsf{F}_\lambda(k,\cdot)$ with a randomly sampled key $k \in \mathcal{K}_\lambda$ or a randomly chosen function $H\colon \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$. Here, $\mathcal{A}$ is in particular able to perform *adaptive* queries, i.e., it can query the oracle on inputs that depend on previously obtained outputs.

Definition 6 (Pseudorandom Function). *An ensemble of PPT functions* $\{\mathsf{F}_\lambda\colon \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ *is a family of* pseudorandom functions*, if for all oracle PPT adversaries* $\mathcal{A}$ *and all large enough* $\lambda \in \mathbb{N}$*, we have*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A}}(\lambda) := \Bigg| \Big[&\Pr\left[\mathcal{A}^{\mathsf{F}_\lambda(k,\cdot)}(1^\lambda) = 1 \mid k \in_R \mathcal{K}_\lambda\right] \\ &- \Pr\left[\mathcal{A}^{H(\cdot)}(1^\lambda) = 1 \mid H \in_R \mathsf{Funs}[\mathcal{X}_\lambda \to \mathcal{Y}_\lambda]\right]\Big]\Bigg| \le \mathsf{negl}(\lambda).\end{aligned}$$

A weaker notion is the *non-adaptive PRF*, where $\mathcal{A}$ must decide on all oracle queries before learning any of the corresponding outputs (see e.g. [MP04; Mye04; BH12; ST20]). Miao et al. [Mia+20] similarly defined *ℓ-selective security*, where $\mathcal{A}$ specifies $\ell+1$ inputs $x_1,\ldots,x_{\ell+1} \in \mathcal{X}_\lambda$. Then it receives PRF outputs $\mathsf{F}_\lambda(k,x_i)$ for the first ℓ inputs under a randomly chosen key $k \in_R \mathcal{K}_\lambda$, and either $\mathsf{F}_\lambda(k,x_{\ell+1})$ or a uniformly random $h \in_R \mathcal{Y}_\lambda$.

Definition 7 (ℓ-Selectively Secure PRF [Mia+20][3]). *For an ensemble of PPT functions* $\{\mathsf{F}_\lambda\colon \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ *define the following game* $\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},b}(\lambda)$ *for a two-stage adversary* $\mathcal{A} = (\mathcal{A}_1,\mathcal{A}_2)$ *and* $b \in \{\mathsf{real},\mathsf{rand}\}$*:*

1. *Run* $(\mathsf{st},(x_i)_{i\in[\ell+1]}) \leftarrow \mathcal{A}_1(1^\lambda)$ *such that all* x_i *are pairwise distinct.*
2. *Sample* $k \in_R \mathcal{K}_\lambda$*.*
3. *Compute* $y_i := \mathsf{F}_\lambda(k,x_i)$ *for* $i \in [\ell]$*.*
4. *Set* $y^{(\mathsf{rand})}_{\ell+1} \in_R \mathcal{Y}_\lambda$ *and* $y^{(\mathsf{real})}_{\ell+1} := \mathsf{F}_\lambda(k,x_{\ell+1})$*.*
5. *Output* $\mathcal{A}_2(1^\lambda,\mathsf{st},(y_i)_{i\in[\ell]},y^{(b)}_{\ell+1})$*.*

We say $\{\mathsf{F}_\lambda\}_\lambda$ *is a family of* ℓ-selectively secure pseudorandom functions*, if for all PPT adversaries* $\mathcal{A}$ *and all large enough* $\lambda \in \mathbb{N}$*, we have*

$$\mathsf{Adv}^{\ell\text{-naPRF}}_{\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{Game}^{\ell\text{-ssPRF}}_{\mathcal{A},\mathsf{real}}(\lambda)\right] - \Pr\left[\mathsf{Game}^{\ell\text{-ssPRF}}_{\mathcal{A},\mathsf{rand}}(\lambda)\right]\right| \le \mathsf{negl}(\lambda). \tag{19}$$

Note that this definition differs from the definition of a selectively secure PRF by Hemenway et al. [Hem+16]. Here, the adversary $\mathcal{A}$ first choses a challenge point $x^* \in \mathcal{X}_\lambda$ and obtains either $y^* = \mathsf{F}_\lambda(k,x^*)$ or $y^* \in_R \mathcal{Y}_\lambda$. Then is can (adaptively) query $\mathsf{F}_\lambda(k,\cdot)$ on any point $x \neq x^*$.

In Lemma 3, we will prove that ℓ-non-adaptive security is implied by the notion of [Mia+20]:

Lemma 3. *An* $(\ell-1)$*-selectively secure family of pseudorandom functions* $\{\mathsf{F}_\lambda\}_\lambda$ *(Definition 7) is also* ℓ*-non-adaptively secure (Definition 7). Every adversary* $\mathcal{A}$ *with advantage* $\mathsf{Adv}^{\ell\text{-naPRF}}_{\mathcal{A}}(\lambda)$ *for the latter, can be transformed into an adversary* $\mathcal{B}$ *for the former with advantage* $\mathsf{Adv}^{(\ell-1)\text{-ssPRF}}_{\mathcal{B}}(\lambda) = \frac{1}{\ell}\cdot\mathsf{Adv}^{\ell\text{-naPRF}}_{\mathcal{A}}(\lambda)$*.*

Proof. We prove the statement using a standard hybrid argument. Assume towards contradiction that there exists an adversary $\mathcal{A}$ with non-negligible advantage $\mathsf{Adv}^{\ell\text{-naPRF}}_{\mathcal{A}}(\lambda)$. Using $\mathcal{A}$ as a black box, we construct a new adversary $\mathcal{B}$.

Define hybrid games $\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},i}(\lambda)$ for $i \in [0,\ell]$ as follows:

1. Run $(\mathsf{st},x_1,\ldots,x_\ell) \leftarrow \mathcal{A}_1(1^\lambda)$.
2. Sample $k \in_R \mathcal{K}_\lambda$ and $H \in_R \mathsf{Funs}[\mathcal{X} \to \mathcal{Y}]$.
3. Output $\mathcal{A}_2(1^\lambda,\mathsf{st},(\mathsf{F}(k,x_j))_{j\le i},(H(x_j))_{j>i})$.

[3] Actually the definition of selective security in [Mia+20] is broken: For any candidate selectively secure PRF F, an adversary can win the security game with good probability by choosing $x_1 = \cdots = x_{\ell+1}$ and checking whether the last output $y_{\ell+1}$ equals the first ℓ outputs. This is always the case if $y_{\ell+1} = \mathsf{F}(k,x_{\ell+1})$, but happens only with probability at most $1/|\mathcal{Y}|$ if $y_{\ell+1} \in_R \mathcal{Y}$. The definition is easily fixed by requiring that all the x_i are different. This is already implicitly assumed by the security proof for their construction.

Note that we have $\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},\mathsf{real}} = \mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},\ell}$ and $\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},\mathsf{rand}} = \mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},0}$.

Now we define the adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ for $\mathsf{Game}^{(\ell-1)\text{-ssPRF}}_{\mathcal{A},b}$: In the first stage ($\mathcal{B}_1$), we run $(\mathsf{st}, x_1, \ldots, x_\ell) \leftarrow \mathcal{A}_1(1^\lambda)$ (for simplicity, assume that all the x_i are distinct), and sample $i^* \in_R [\ell]$. Now the goal is to provide $\mathcal{A}_2$ with a view that matches its view in either $\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},i^*-1}(\lambda)$ or $\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},i^*}(\lambda)$. So we need to pass $\mathsf{F}_\lambda(k, x_j)$ for $j \in [i^*-1]$, either $\mathsf{F}_\lambda(k, x_{i^*})$ or random $h_{i^*} \in_R \mathcal{Y}_\lambda$ at index i^*, and random $h_j \in_R \mathcal{Y}_\lambda$ for $j \in [i^*+1, \ell]$. Therefore, we output $((\mathsf{st}, i^*), x_1, \ldots, x_{i^*-1}, x_{i^*+1}, \ldots, x_\ell, x_{i^*})$. Note that we moved the element x_{i^*} to the end of the tuple. In the second stage ($\mathcal{B}_2$), we get as input from the challenger $((\mathsf{st}, i^*), y_1, \ldots, y_{i^*-1}, y_{i^*+1}, \ldots, y_\ell, y_{i^*})$ where $y_j = \mathsf{F}_\lambda(k, x_j)$ for all $j \in [\ell] \setminus \{i^*\}$ and either $y_{i^*} = \mathsf{F}_\lambda(k, x_{i^*})$ (if $b = \mathsf{real}$) or $y_{i^*} \in_R \mathcal{Y}_\lambda$ (if $b = \mathsf{rand}$). We sample random $h_j \in_R \mathcal{Y}_\lambda$ for $j \in [i^*+1, \ell]$, and run $\mathcal{A}_2(1^\lambda, \mathsf{st}, y_1, \ldots, y_{i^*}, h_{i^*+1}, \ldots, h_\ell)$. The output of $\mathcal{B}_2$ is whatever $\mathcal{A}_2$ outputs. Hence, we have

$$\begin{aligned}
\Pr\left[\mathsf{Game}^{(\ell-1)\text{-ssPRF}}_{\mathcal{B},\mathsf{real}}(\lambda) = 1\right] &= \sum_{j\in[\ell]} \Pr[i^* = j] \cdot \Pr\left[\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},i^*}(\lambda) = 1 \;\middle|\; i^* = j\right] \\
&= \frac{1}{\ell} \sum_{j\in[\ell]} \Pr\left[\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},j}(\lambda) = 1\right], \text{ and} \\
\Pr\left[\mathsf{Game}^{(\ell-1)\text{-ssPRF}}_{\mathcal{B},\mathsf{rand}}(\lambda) = 1\right] &= \sum_{j\in[\ell]} \Pr[i^* = j] \cdot \Pr\left[\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},i^*-1}(\lambda) = 1 \;\middle|\; i^* = j\right] \\
&= \frac{1}{\ell} \sum_{j\in[0,\ell-1]} \Pr\left[\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},j}(\lambda) = 1\right].
\end{aligned}$$

When plugging the above into Equation (19) all but the terms for $j = 0$ and $j = \ell$ cancel out, and we obtain the advantage of $\mathcal{B}$ in the $(\ell-1)$-ssPRF game, which concludes the proof:

$$\begin{aligned}
\mathsf{Adv}^{(\ell-1)\text{-ssPRF}}_{\mathcal{B}}(\lambda) &= \frac{1}{\ell} \cdot \left|\Pr\left[\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},\ell}(\lambda) = 1\right] - \Pr\left[\mathsf{Game}^{\ell\text{-naPRF}}_{\mathcal{A},0}(\lambda) = 1\right]\right| \\
&\overset{(1)}{=} \frac{1}{\ell} \cdot \mathsf{Adv}^{\ell\text{-naPRF}}_{\mathcal{A}}(\lambda).
\end{aligned}$$

□

Definition 8 (Computational ℓ-DHI Assumption (following [DY05])). *Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q. We say that the ℓ-DHI assumption holds for $\mathbb{G}$ if for every PPT algorithm $\mathcal{A}$ we have*

$$\Pr\left[\mathcal{A}(1^\lambda, g, g^x, \ldots, g^{x^\ell}) = g^{\frac{1}{x}} \mid x \in_R \mathbb{F}_q^*\right] \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random choice of x and the randomness used by $\mathcal{A}$.

C Helper Functionalities

Server Broadcast Functionality $\mathcal{F}_{\mathsf{SBCast}}$

Broadcast On input $(\mathsf{Broadcast}, m)$ from S_j, $j \in \{1,2\}$, $\mathcal{F}_{\mathsf{SBCast}}$ sends $(\mathsf{BroadcastMsg}, \mathsf{S}_j, m)$ to the other server and all clients.

Fig. 7: The server broadcast functionality $\mathcal{F}_{\mathsf{SBCast}}$

Commitment Functionality $\mathcal{F}_{\mathsf{Com}}$

Commit On input $(\mathsf{Commit}, \mathsf{id}, m)$ from S_j, $j \in \{1,2\}$, where id is a fresh identifier, $\mathcal{F}_{\mathsf{Com}}$ stores $(\mathsf{S}_j, \mathsf{id}, m)$ and sends $(\mathsf{Committed}, \mathsf{S}_j, \mathsf{id})$ to the other server.
Open On input $(\mathsf{Open}, \mathsf{id})$ from S_j, $j \in \{1,2\}$, if $\mathcal{F}_{\mathsf{Com}}$ has stored a tuple $(\mathsf{S}_j, \mathsf{id}, m)$, it sends $(\mathsf{Opened}, \mathsf{S}_j, \mathsf{id}, m)$ to the other server.

Fig. 8: The commitment functionality $\mathcal{F}_{\mathsf{Com}}$

Coin Tossing Functionality $\mathcal{F}_{\mathsf{Rand}}$

Rand On input $(\mathsf{Rand}, n, p \in [0,1])$ from S_1 and S_2, $\mathcal{F}_{\mathsf{Rand}}$ samples a string $r \in \{0,1\}^n$ such that $r_i \leftarrow \mathsf{Ber}(p)$ for $i \in [n]$, and returns r to both servers.

Fig. 9: The coin tossing functionality $\mathcal{F}_{\mathsf{Rand}}$

D Further Zero-Knowledge Proofs

We keep the public parameters and public keys implicit except for the proofs involving shares of the public and secret keys, e.g., for partial decryptions.

At two places in Π_{Enc}, we need straight-line extractable proofs, and denote them with $\Pi^{\mathsf{ZK\text{-}SE}}$. They can be instantiated by applying the Fischlin transformation [Fis05] to the canonical Σ protocol. For example, $\Pi^{\mathsf{ZK\text{-}SE}}_{\mathsf{CL\text{-}PoPK}}$ is the straight-line extractable version of $\Pi^{\mathsf{ZK}}_{\mathsf{CL\text{-}PoPK}}$.

Discrete Logarithm The standard proofs for the discrete logarithm in the CL group ($\Pi^{\mathsf{ZK}}_{\mathsf{CL\text{-}DLog}}$) and in the prime order groups ($\Pi^{\mathsf{ZK}}_{\mathsf{PO\text{-}DLog}}$) are building blocks for several of the following proofs.

$$\mathsf{R}^{\mathsf{cl}}_{\mathsf{DLog}} := \left\{h; x \;\middle|\; h = g_q^x\right\} \tag{20}$$
$$\mathsf{R}^{\mathsf{po}}_{\mathsf{DLog}} := \{h; x \mid h = g^x\} \tag{21}$$

Key Generation When generating the public key shares in Π_{Enc}, the servers need to prove that they are well-formed. We use a straight-line extractable PoK $\Pi^{\mathsf{ZK\text{-}SE}}_{\mathsf{ServerPK}}$ here to allow the simulator to extract the secret key (share) of the corrupted server.

$$\mathsf{R}_{\mathsf{ServerPK}} := \left\{(\mathsf{clpk}_j, \mathsf{popk}_j); (\mathsf{clsk}_j, \mathsf{posk}_j) \;\middle|\; (\mathsf{clpk}_j; \mathsf{clsk}_j) \in \mathsf{R}^{\mathsf{cl}}_{\mathsf{DLog}} \wedge (\mathsf{popk}_j; \mathsf{posk}_j) \in \mathsf{R}^{\mathsf{po}}_{\mathsf{DLog}}\right\} \tag{22}$$

Partial Decryptions To realize the threshold decryption functionality, we need to use proofs of partial decryption for HSM-CL encryption ($\Pi^{\mathsf{ZK}}_{\mathsf{CL\text{-}PoPD}}$) and ElGamal ($\Pi^{\mathsf{ZK}}_{\mathsf{PO\text{-}PoPD}}$) for the following relations:

$$\begin{aligned}\mathsf{R}^{\mathsf{cl}}_{\mathsf{PoPD}} := \Big\{&(\mathsf{clpk}, \mathsf{clct}, \mathsf{clct}'); (\mathsf{clsk}) \\ &\Big|\; (\mathsf{clpk}; \mathsf{clsk}) \in \mathsf{R}^{\mathsf{cl}}_{\mathsf{DLog}} \wedge {\mathsf{clct}_1}' = \mathsf{clct}_1 \wedge {\mathsf{clct}_2}' = \mathsf{clct}_2 \cdot (\mathsf{clct}_1)^{-\mathsf{clsk}}\Big\}\end{aligned} \tag{23}$$

$$\begin{aligned}\mathsf{R}^{\mathsf{po}}_{\mathsf{PoPD}} := \Big\{&(\mathsf{popk}, \mathsf{poct}, \mathsf{poct}'); (\mathsf{posk}) \\ &\Big|\; (\mathsf{popk}; \mathsf{posk}) \in \mathsf{R}^{\mathsf{po}}_{\mathsf{DLog}} \wedge {\mathsf{poct}_1}' = \mathsf{poct}_1 \wedge {\mathsf{poct}_2}' = \mathsf{poct}_2 \cdot (\mathsf{poct}_1)^{-\mathsf{clsk}}\Big\}\end{aligned} \tag{24}$$

Moreover, we also need a proof $\Pi^{\mathsf{ZK}}_{\mathsf{DecExpInv}}$ that we obtained a $\mathbb{G}$ element by raising the generator g to the inverse of whatever we just decrypted.

$$\begin{aligned}\mathsf{R}_{\mathsf{DecExpInv}} := \Big\{&(\mathsf{clpk}, \mathsf{clct}, h); (\mathsf{clsk}, m) \\ &\Big|\; (\mathsf{clpk}; \mathsf{clsk}) \in \mathsf{R}^{\mathsf{cl}}_{\mathsf{DLog}} \wedge f^m = \mathsf{clct}_2 \cdot (\mathsf{clct}_1)^{-\mathsf{clsk}} \wedge h^m = g\Big\}\end{aligned} \tag{25}$$

Rerandomization of Ciphertexts The following two relations formalize the rerandomization of HSM-CL and ElGamal ciphertexts.

$$\begin{aligned}\mathsf{R}^{\mathsf{cl}}_{\mathsf{Randomize}} := \Big\{&(\mathsf{clct}, \mathsf{clct}'); (r) \\ &\Big|\; \mathsf{clct}'_1 = \mathsf{clct}_1 \cdot g_q^r \wedge \mathsf{clct}'_2 = \mathsf{clct}_2 \cdot \mathsf{clpk}^r\Big\}\end{aligned} \tag{26}$$

$$\begin{aligned}\mathsf{R}^{\mathsf{po}}_{\mathsf{Randomize}} := \Big\{&(\mathsf{poct}, \mathsf{poct}'); (r) \\ &\Big|\; \mathsf{poct}'_1 = \mathsf{poct}_1 \cdot g^r \wedge \mathsf{poct}'_2 = \mathsf{poct}_2 \cdot \mathsf{popk}^r\Big\}\end{aligned} \tag{27}$$

E Two-Party Threshold Encryption

E.1 Ideal Functionality $\mathcal{F}_{\mathsf{Enc}}$

Ideal Encryption Functionality $\mathcal{F}_{\mathsf{Enc}}$

KeyGen On input (KeyGen) from S_j, $j \in \{1,2\}$, $\mathcal{F}_{\mathsf{Enc}}$ runs:

1. $(\mathsf{clsk}, \mathsf{clpk}) \leftarrow \mathsf{KeyGen}_b^{\mathsf{cl}}$, and $(\mathsf{posk}, \mathsf{popk}) \leftarrow \mathsf{KeyGen}_b^{\mathsf{po}}$
2. If one of S_1 and S_2 is corrupted, send $(\mathsf{Leakage}, \mathsf{clpk}, \mathsf{popk})$ to $\mathcal{S}$. Receive $\mathsf{continue}$ or abort from $\mathcal{S}$. In the latter case, abort.
3. Send $(\mathsf{clpk}, \mathsf{popk})$ to all honest S_j and C_i

CL-Random On input (Random) from S_j, $j \in \{1,2\}$, $\mathcal{F}_{\mathsf{Enc}}$ runs:

1. Sample $m \in_R \mathbb{F}_q$ and compute $\mathsf{clct}\text{-}m \leftarrow \mathsf{Enc}^{\mathsf{cl}}(\mathsf{clpk}, m)$.
2. If one of S_1 and S_2 is corrupted, send $(\mathsf{Leakage}, \mathsf{clct}\text{-}m)$ to $\mathcal{S}$. Receive $\mathsf{continue}$ or abort from $\mathcal{S}$. In the latter case, abort.
3. Send $(\mathsf{Output}, \mathsf{clct}\text{-}m)$ to the honest S_j.

CL-Constant On input $(\mathsf{CL\text{-}Constant}, c)$ from S_j, $j \in \{1,2\}$, $\mathcal{F}_{\mathsf{Enc}}$ returns $\mathsf{clct}\text{-}c := \mathsf{Enc}^{\mathsf{cl}}(\mathsf{clpk}, c; 0)$.

PO-Zero On input $(\mathsf{PO\text{-}Zero})$ from S_j, $j \in \{1,2\}$, $\mathcal{F}_{\mathsf{Enc}}$ returns $\mathsf{poct}\text{-}0 := \mathsf{Enc}^{\mathsf{po}}(\mathsf{popk}, 0; 0)$.

CL-Decrypt On input $(\mathsf{CL\text{-}Decrypt}, [\![\mathsf{clct}\text{-}m_1, \ldots, \mathsf{clct}\text{-}m_n]\!])$ from S_j, $j \in \{1,2\}$, $\mathcal{F}_{\mathsf{Enc}}$ runs:

1. Compute $M_i := \mathsf{clct}\text{-}m_{i2} \cdot \mathsf{clct}\text{-}m_{i1}^{-\mathsf{clsk}}$ for $i \in [n]$.
2. If S_j is corrupted, send $(\mathsf{Leakage}, [\![M_1, \ldots, M_n]\!])$ to $\mathcal{S}$. Receive $\mathsf{continue}$ or abort from $\mathcal{S}$. In the latter case, abort.
3. Let $m_i \leftarrow \log_f(M_i)$ for $i \in [n]$. If any $m_i = \perp$, abort. Send $[\![m_1, \ldots, m_n]\!]$ to the honest S_j.

PO-Decrypt On input $(\mathsf{PO\text{-}Decrypt}, [\![\mathsf{poct}\text{-}m_1, \ldots, \mathsf{poct}\text{-}m_n]\!])$ from S_j, $j \in \{1,2\}$, $\mathcal{F}_{\mathsf{Enc}}$ runs:

1. Compute $M_i := \mathsf{poct}\text{-}m_{i2} \cdot \mathsf{poct}\text{-}m_{i1}^{-\mathsf{posk}}$ for $i \in [n]$.
2. If S_j is corrupted, send $(\mathsf{Leakage}, [\![M_1, \ldots, M_n]\!])$ to $\mathcal{S}$. Receive $\mathsf{continue}$ or abort from $\mathcal{S}$. In the latter case, abort.
3. Let $m_i \leftarrow \log_f(M_i)$ (with time limit) for $i \in [n]$. If any $m_i = \perp$, abort. Send $[\![m_1, \ldots, m_n]\!]$ to the honest S_j.

DecExpInv On input $(\mathsf{DecExpInv}, [\![\mathsf{clct}\text{-}m_1, \ldots, \mathsf{clct}\text{-}m_n]\!])$ from S_j, $j \in \{1,2\}$, $\mathcal{F}_{\mathsf{Enc}}$ runs:

1. If S_2 is corrupted, then receive either $\mathsf{continue}$ or abort from $\mathcal{S}$. In the latter case, abort.
2. Compute $M_i \leftarrow \mathsf{clct}\text{-}m_{i2} \cdot \mathsf{clct}\text{-}m_{i1}^{-\mathsf{clsk}}$ for $i \in [n]$.
3. If S_1 is corrupted, then send $(\mathsf{Leakage}, [\![M_1, \ldots, M_n]\!])$ to $\mathcal{S}$. Receive either $\mathsf{continue}$ or abort from $\mathcal{S}$. In the latter case, abort.
4. Let $m_i := \log_f(M_i)$ for $i \in [n]$. If any $m_i = \perp$, abort. Set $h_i := g^{\frac{1}{m_i}}$, and send $[\![h_1, \ldots, h_n]\!]$ to the honest S_j.

Fig. 10: Helper functionality for 2-out-of-2 threshold encryption for CL and ElGamal encryption.

E.2 Protocol $\mathit{\Pi}_{\mathsf{Enc}}$

Protocol $\mathit{\Pi}_{\mathsf{Enc}}$ realizing $\mathcal{F}_{\mathsf{Enc}}$

KeyGen 1. Each S_j

(a) runs $(\mathsf{clsk}_j, \mathsf{clpk}_j) \leftarrow \mathsf{KeyGen}_b^{\mathsf{cl}}(\mathsf{pp}_{\mathsf{cl}})$, $(\mathsf{posk}_j, \mathsf{popk}_j) \leftarrow \mathsf{KeyGen}_b^{\mathsf{po}}(\mathsf{pp}_{\mathsf{po}})$, and computes $\pi_{S,j} \leftarrow \mathit{\Pi}_{\mathsf{ServerPK}}^{\mathsf{ZK\text{-}SE}}.\mathsf{Prove}\big((\mathsf{clpk}_j, \mathsf{popk}_j), (\mathsf{clsk}_j, \mathsf{posk}_j)\big)$.

(b) send $(\mathsf{Commit}, (\mathsf{clpk}_j, \mathsf{popk}_j, \pi_{S,j}))$ to $\mathcal{F}_{\mathsf{Com}}$

(c) once received $(\mathsf{Committed}, \mathsf{S}_{3-j})$ from $\mathcal{F}_{\mathsf{Com}}$, send send (Open) to $\mathcal{F}_{\mathsf{Com}}$

(d) receive $(\mathsf{Open}, \mathsf{S}_{3-j}, (\mathsf{clpk}_{3-j}, \mathsf{popk}_{3-j}, \pi_{S,3-j}))$ from $\mathcal{F}_{\mathsf{Com}}$

(e) abort if $\mathit{\Pi}_{\mathsf{ServerPK}}^{\mathsf{ZK\text{-}SE}}.\mathsf{Verify}\big(\pi_{S,3-j}, (\mathsf{clpk}_{3-j}, \mathsf{popk}_{3-j})\big) = \perp$.

(f) set $\mathsf{clpk} := \mathsf{clpk}_1 \cdot \mathsf{clpk}_2$ and $\mathsf{popk} := \mathsf{popk}_1 \cdot \mathsf{popk}_2$ (implicitly $\mathsf{clsk} := \mathsf{clsk}_1 + \mathsf{clsk}_2$ and $\mathsf{posk} := \mathsf{posk}_1 + \mathsf{posk}_2$).

(g) and sends $(\mathsf{Broadcast}, (\mathsf{clpk}, \mathsf{popk}))$ to $\mathcal{F}_{\mathsf{SBCast}}$

(h) abort if the public keys broadcasted by S_{3-j} do not equal the computed ones

2. Each C_i aborts if the broadcasted public keys are not equal, otherwise it stores the public keys $(\mathsf{clpk}, \mathsf{popk})$

CL-Random On input (Random), each S_j does

1. sample $m_j \in_R \mathbb{F}_q$, and compute $\mathsf{clct}\text{-}m_j \leftarrow \mathsf{Enc}^{\mathsf{cl}}(\mathsf{clpk}, m_j; r_j)$ and $\pi_{R,j} \leftarrow \Pi^{\mathsf{ZK\text{-}SE}}_{\mathsf{CL\text{-}PoPK}}.\mathsf{Prove}(\mathsf{clct}\text{-}m_j, (m_j, r_j))$.
2. send $(\mathsf{Commit}, (\mathsf{clct}\text{-}m_j, \pi_{R,j}))$ to $\mathcal{F}_{\mathsf{Com}}$
3. once received $(\mathsf{Committed}, \mathsf{S}_{3-j})$ from $\mathcal{F}_{\mathsf{Com}}$, send (Open) to $\mathcal{F}_{\mathsf{Com}}$
4. receive $(\mathsf{Open}, \mathsf{S}_{3-j}, (\mathsf{clct}\text{-}m_{3-j}, \pi_{R,3-j}))$ from $\mathcal{F}_{\mathsf{Com}}$
5. abort if $\Pi^{\mathsf{ZK\text{-}SE}}_{\mathsf{CL\text{-}PoPK}}.\mathsf{Verify}(\pi_{R,3-j}, \mathsf{clct}\text{-}m_{3-j}) = \perp$.
6. set $\mathsf{clct}\text{-}m := \mathsf{clct}\text{-}m_1 + \mathsf{clct}\text{-}m_2$
7. and sends $(\mathsf{Broadcast}, \mathsf{clct}\text{-}m)$ to $\mathcal{F}_{\mathsf{SBCast}}$
8. abort if the ciphertext broadcasted by S_{3-j} do not equal the computed $\mathsf{clct}\text{-}m$

Clients abort the servers broadcast different ciphertexts.

CL-Constant On input $(\mathsf{CL\text{-}Constant}, c)$ each S_j outputs $\mathsf{clct}\text{-}c := \mathsf{Enc}^{\mathsf{cl}}(\mathsf{clpk}, c; 0)$.

PO-Zero On input $(\mathsf{PO\text{-}Zero})$ each S_j outputs $\mathsf{poct}\text{-}0 := \mathsf{Enc}^{\mathsf{po}}(\mathsf{popk}, 0; 0)$.

CL-Decrypt On input $(\mathsf{CL\text{-}Decrypt}, [\![\mathsf{clct}\text{-}m_1, \ldots, \mathsf{clct}\text{-}m_n]\!])$, each S_j does

1. Compute $\mathsf{clct}\text{-}{m_i}^{(j)} := \mathsf{PDec}^{\mathsf{cl}}(\mathsf{clsk}_j, \mathsf{clct}\text{-}m_i)$ and $\pi_{D,i,j} \leftarrow \Pi^{\mathsf{ZK}}_{\mathsf{CL\text{-}PoPD}}.\mathsf{Prove}((\mathsf{clpk}_j, \mathsf{clct}\text{-}m_i, \mathsf{clct}\text{-}{m_i}^{(j)}), \mathsf{clsk}_j)$ for $i \in [n]$.
2. Send $(\mathsf{clct}\text{-}{m_{i2}}^{(j)}, \pi_{D,i,j})_{i\in[n]}$ to S_{3-j}.
3. Abort if $\Pi^{\mathsf{ZK}}_{\mathsf{CL\text{-}PoPD}}.\mathsf{Verify}(\pi_{D,i,3-j}, (\mathsf{clpk}_{3-j}, \mathsf{clct}\text{-}m_i, \mathsf{clct}\text{-}{m_i}^{(3-j)}))$ for any $i \in [n]$.
4. Compute $m_i := \mathsf{Dec}^{\mathsf{cl}}(\mathsf{clsk}_j, \mathsf{clct}\text{-}{m_i}^{(3-j)})$. If any $m_i = \perp$, abort. Otherwise, output $[\![m_1, \ldots, m_n]\!]$.

PO-Decrypt Analogous to CL-Decrypt.

DecExplnv On input $(\mathsf{DecExplnv}, [\![\mathsf{clct}\text{-}m_1, \ldots, \mathsf{clct}\text{-}m_n]\!])$

1. S_2 computes $\mathsf{clct}\text{-}{m_i}' \leftarrow \mathsf{PDec}^{\mathsf{cl}}(\mathsf{clsk}_2, \mathsf{clct}\text{-}m_i)$ for $i \in [n]$ and $\pi_{DEI,i,2} \leftarrow \Pi^{\mathsf{ZK}}_{\mathsf{CL\text{-}PoPD}}.\mathsf{Prove}((\mathsf{clpk}_2, \mathsf{clct}\text{-}m_i, \mathsf{clct}\text{-}{m_i}'), \mathsf{clsk}_2)$ and sends $(\mathsf{clct}\text{-}m'_{i2}, \pi_{DEI,i,2})$ (NB: $\mathsf{clct}\text{-}m'_{i1} = \mathsf{clct}\text{-}m_{i1}$) for $i \in [n]$ to S_1
2. S_1 aborts if $\Pi^{\mathsf{ZK}}_{\mathsf{CL\text{-}PoPD}}.\mathsf{Verify}(\pi_{DEI,i,2}, (\mathsf{clpk}_2, \mathsf{clct}\text{-}m_i, \mathsf{clct}\text{-}{m_i}')) = \perp$ for any $i \in [n]$. Otherwise it computes $m_i \leftarrow \mathsf{Dec}^{\mathsf{cl}}(\mathsf{clsk}_1, \mathsf{clct}\text{-}{m_i}')$ for $i \in [n]$. If $m_i = \perp$ for any i, abort. Send $h_i := g^{\frac{1}{m_i}}$ and $\pi_{DEI,i,1} \leftarrow \Pi^{\mathsf{ZK}}_{\mathsf{DecExplnv}}.\mathsf{Prove}((\mathsf{clpk}_2, \mathsf{clct}\text{-}{m_i}', h_i), \mathsf{clsk}_1)$ for $i \in [n]$ to S_2
3. S_2 aborts if $\Pi^{\mathsf{ZK}}_{\mathsf{DecExplnv}}.\mathsf{Verify}(\pi_{DEI,i,1}, (\mathsf{clpk}_2, \mathsf{clct}\text{-}{m_i}', h_i))$ for any $i \in [n]$.
4. Both parties output $[\![h_1, \ldots, h_n]\!]$.

Fig. 11: Encryption helper protocol in the $(\mathcal{F}_{\mathsf{SBCast}}, \mathcal{F}_{\mathsf{Com}})$-hybrid model

E.3 Proof of Security

Theorem 8. *The protocol Π_{Enc} (Figure 11) securely realizes the functionality $\mathcal{F}_{\mathsf{Enc}}$ (Figure10) in the $(\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{SBCast}})$-hybrid model with computational security, tolerating active and static corruptions of any number of clients C_i and at most one server S_j.*

Proof. We give the simulator $\mathcal{S}_{\mathsf{Enc}}$ in Figure 12. In the following, we argue that the simulation is indistinguishable to the real execution, because for each method $\mathcal{S}_{\mathsf{Hist}}$ can send what the adversary would expect to see given their state and the output generated by $\mathcal{F}_{\mathsf{Enc}}$.

KeyGen $\mathcal{S}_{\mathsf{Enc}}$ extracts the public key shares $(\mathsf{clpk}_{\mathsf{c}}, \mathsf{popk}_{\mathsf{c}})$ chosen by the corrupted server from the commitment, and also extracts the corresponding secret keys from the straight-line extractable proof. It then computes shares $(\mathsf{clpk}_{\mathsf{h}}, \mathsf{popk}_{\mathsf{h}})$ such that the products match the pair of public keys $(\mathsf{clpk}, \mathsf{popk})$ received from $\mathcal{F}_{\mathsf{Enc}}$. Since $\mathcal{S}_{\mathsf{Hist}}$ does not know the corresponding secret keys $(\mathsf{clsk}_{\mathsf{h}}, \mathsf{posk}_{\mathsf{h}})$, it simulates the proofs.

CL-Random This method is simulated analogously to the key generation. $\mathcal{S}_{\mathsf{Enc}}$ extracts the ciphertext prepared by the server, and then computes a ciphertext on behalf of the honest party such that their product matches the output of $\mathcal{F}_{\mathsf{Enc}}$.

CL-Decrypt, PO-Decrypt, and DecExpInv Here, $\mathcal{S}_{\mathsf{Enc}}$ obtains the decrypted values M_i from $\mathcal{F}_{\mathsf{Enc}}$. From these values, the input ciphertexts, and the corrupted server's share of the secret key, it can exactly compute what the honest server would have to send without having to know their share of the secret key. □

Simulator for $\mathit{\Pi}_{\mathsf{Enc}}$ realizing $\mathcal{F}_{\mathsf{Enc}}$

KeyGen Case of $\mathsf{S_h}$ being honest and $\mathsf{S_c}$ being corrupted:

1. After receiving $(\mathsf{KeyGenReceived}, \mathsf{S_h})$ from $\mathcal{F}_{\mathsf{Enc}}$, send $(\mathsf{Committed}, \mathsf{S_h})$ from $\mathcal{F}_{\mathsf{Com}}$ to $\mathsf{S_c}$.
2. Receive a message $(\mathsf{Commit}, (\mathsf{clpk_c}, \mathsf{popk_c}, \pi_{S,\mathsf{c}}))$ for $\mathcal{F}_{\mathsf{Com}}$ from $\mathsf{S_c}$.
3. Verify $\mathit{\Pi}^{\mathsf{ZK\text{-}SE}}_{\mathsf{ServerPK}}.\mathsf{Verify}(\pi_{S,\mathsf{c}}, (\mathsf{clpk_c}, \mathsf{popk_c})) \in \{\perp, \top\}$. If successful, extract $\mathsf{clsk_c}$ and $\mathsf{posk_c}$ from $\pi_{S,\mathsf{c}}$ and store them for later. Otherwise, set $\mathsf{clpk_c} = 1$ and $\mathsf{popk_c} = 1$ (in the corresponding groups).
4. Send (KeyGen) on behalf of $\mathsf{S_c}$ to $\mathcal{F}_{\mathsf{Enc}}$.
5. Receive $(\mathsf{Leakage}, \mathsf{clpk}, \mathsf{popk})$ from $\mathcal{F}_{\mathsf{Enc}}$.
6. Set $\mathsf{clpk_h} := \mathsf{clpk} \cdot \mathsf{clpk_c}^{-1}$ and $\mathsf{popk_h} := \mathsf{popk} \cdot \mathsf{popk_c}^{-1}$. Simulate a corresponding proof $\pi_{S,\mathsf{h}} \leftarrow \mathit{\Pi}^{\mathsf{ZK\text{-}SE}}_{\mathsf{ServerPK}}.\mathsf{SimProve}((\mathsf{clpk_h}, \mathsf{popk_h}))$.
7. Send $(\mathsf{Open}, \mathsf{S_h}, (\mathsf{clpk_h}, \mathsf{popk_h}, \pi_{S,\mathsf{h}}))$ from $\mathcal{F}_{\mathsf{Com}}$ to $\mathsf{S_c}$.
8. Receive (Open) from $\mathcal{F}_{\mathsf{Com}}$ from $\mathsf{S_c}$.
9. Simulate abort of $\mathsf{S_h}$ if $\pi_{S,\mathsf{c}}$ was invalid.
10. Simulate Steps 1g to 2 for $\mathsf{S_h}$ and honest C_i. If any of them aborts, send (abort) to $\mathcal{F}_{\mathsf{Enc}}$ and stop. Otherwise, send $(\mathsf{continue})$.

Case of both $\mathsf{S_1}$ and $\mathsf{S_2}$ honest: Perform the simulation where $\mathsf{S_1}$ follows the protocol and the actions of $\mathsf{S_2}$ are simulated in the same way as $\mathsf{S_h}$ in the case above.

CL-Random Case of $\mathsf{S_h}$ being honest and $\mathsf{S_c}$ being corrupted:

1. Simulate in the same way as the KeyGen operation

Case of both $\mathsf{S_1}$ and $\mathsf{S_2}$ honest: Nothing to simulate since the clients do not receive any output.

CL-Constant Nothing to simulate since the parties have only local computation.

PO-Zero Nothing to simulate since the parties have only local computation.

CL-Decrypt Case of $\mathsf{S_h}$ being honest and $\mathsf{S_c}$ being corrupted: – On input $[\![\mathsf{clct}\text{-}m_1, \ldots, \mathsf{clct}\text{-}m_n]\!]$

1. Receive $(\mathsf{Leakage}, [\![M_1, \ldots, M_n]\!])$ from $\mathcal{F}_{\mathsf{Enc}}$.
2. Compute $W_i := \mathsf{clct}\text{-}m_{i2} \cdot \mathsf{clct}\text{-}m_{i1}^{\mathsf{clsk_c}} \cdot M_i^{-1}$ and simulate proofs of correctness $\pi_{D,i,\mathsf{h}} \leftarrow \mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{CL\text{-}PoPD}}.\mathsf{Prove}((\mathsf{clpk_h}, \mathsf{clct}\text{-}m_i, (\mathsf{clct}\text{-}m_{i1}, W_i)))$ for $i \in [n]$.
3. Send $[\![(W_i, \pi_{D,i,\mathsf{h}})]\!]_{i \in [n]}$ on behalf of $\mathsf{S_h}$ to $\mathsf{S_c}$.
4. If $\mathsf{S_c}$ sends no or an invalid response, send abort to $\mathcal{F}_{\mathsf{Enc}}$. Otherwise, send $\mathsf{continue}$.

Case of both $\mathsf{S_1}$ and $\mathsf{S_2}$ honest: Nothing to simulate since the clients do not receive any output.

PO-Decrypt Analogous to CL-Decrypt.

DecExpInv Case of $\mathsf{S_1}$ being honest and $\mathsf{S_2}$ being corrupted:

1. Receive $\mathsf{clct}\text{-}m_{i2}'$ for $i \in [n]$ and proofs of correctness from $\mathsf{S_2}$. If proof verifies, send $\mathsf{continue}$ to $\mathcal{F}_{\mathsf{Enc}}$.
2. If $\mathcal{F}_{\mathsf{Enc}}$ aborts, simulate abort. Otherwise, receive $h_1, \ldots, h_n$ from $\mathcal{F}_{\mathsf{Enc}}$.
3. Simulate proofs of correctness w.r.t. $\mathsf{clct}\text{-}m_i{}'$ and h_i, and send both to $\mathsf{S_2}$.

Case of $\mathsf{S_2}$ being honest and $\mathsf{S_1}$ being corrupted:

1. Receive $(\mathsf{Leakage}, [\![M_1, \ldots, M_n]\!])$ from $\mathcal{F}_{\mathsf{Enc}}$.
2. Compute $\mathsf{clct}\text{-}m_{i2}' := M_i \cdot \mathsf{clct}\text{-}m_{i1}^{\mathsf{clsk_1}}$ for $i \in [n]$, simulate the corresponding proofs, and send all to $\mathsf{S_1}$.
3. Receive $h_1, \ldots, h_n$ and proofs from $\mathsf{S_1}$ (alternatively, if it aborts, send abort to $\mathcal{F}_{\mathsf{Enc}}$). If they verify, send $\mathsf{continue}$ to $\mathcal{F}_{\mathsf{Enc}}$, otherwise send abort.

Case of both $\mathsf{S_1}$ and $\mathsf{S_2}$ honest: Nothing to simulate since the clients do not receive any output.

Fig. 12: Simulator $\mathcal{S}_{\mathsf{Enc}}$ for the security proof $\mathit{\Pi}_{\mathsf{Enc}}$ for Theorem 8.

F Noise Generation Algorithms

Noise Generation Algorithms

The algorithms work either on plaintext or on encrypted tuples depending on the mode parameter. Moreover, when $\mathsf{mode} = \mathsf{enc}$, then clpk is expected to be a valid CL public key, and S consists of ciphertexts under this key. In this case, we also output the necessary witnesses $\mathcal{W}$ to create proofs of plaintext knowledge.

```
$\mathsf{SampleFrequencyDummies}(\mathcal{D}, T, \lambda_3, t_3, \mathsf{mode} \in \{\mathsf{plain}, \mathsf{enc}\}, \mathsf{clpk})$
 1. Let $\mathcal{R} := [\![]\!]$ and $\mathcal{W} := [\![]\!]$ be empty lists.
 2. For every $i \in [T]$:
    (a) sample $N_i \leftarrow \mathsf{TSDLap}(\lambda_3, t_3)$
    (b) for $j \in [N_i]$:
          i. choose fresh $x \in \mathcal{D}$
         ii. – If $\mathsf{mode} = \mathsf{plain}$, then append $i$ copies of $x$ to $\mathcal{R}$.
             – If $\mathsf{mode} = \mathsf{enc}$, then append $\mathsf{Enc}^{\mathsf{cl}}(\mathsf{clpk}, x; r_k)$ for $k \in [i]$ to $\mathcal{R}$ where $r_k$ denotes the
               randomness used in the $k$th encryption.
               Also append $(x, r_1), \ldots, (x, r_i)$ to $\mathcal{W}$.
 3. Output $\mathcal{R}, \mathcal{W}$
$\mathsf{SampleDuplicateDummies}(S = (u_i)_{i \in [n]}, p; \mathsf{rnd} \in \{0,1\}^n)$
 1. Let $\mathcal{R} := [\![]\!]$ be an empty list.
 2. For every $i \in [n]$:
    (a) If $\mathsf{rnd}$ is explicitly passed, set $N_i := \mathsf{rnd}_i$. Otherwise, sample $N_i \leftarrow \mathsf{Ber}(p)$.
    (b) If $N_i = 1$, append $u_i$ to $\mathcal{R}$.
 3. Output $\mathcal{R}$
$\mathsf{SampleDummyBuckets}(\Delta, \lambda_2, t_2, \mathsf{mode} \in \{\mathsf{plain}, \mathsf{enc}\}, \mathsf{popk})$
 1. Let $\mathcal{R} := [\![]\!]$ and $\mathcal{W} := [\![]\!]$ be empty lists.
 2. For every $i \in [\Delta]$:
    (a) sample $N_i \leftarrow \mathsf{TSDLap}(\lambda_2, t_2)$
    (b) for $j \in [N_i]$:
          – If $\mathsf{mode} = \mathsf{plain}$, then append $i$ to $\mathcal{R}$.
          – If $\mathsf{mode} = \mathsf{enc}$, then append $\mathsf{Enc}^{\mathsf{po}}(\mathsf{popk}, i; r_i)$ to $\mathcal{R}$ where $r_i \in \mathbb{F}_q$ denotes the random-
            ness used in the encryption. Also append $(i, r_i)$ to $\mathcal{W}$.
 3. Output $\mathcal{R}$
```

Fig. 13: Algorithms to generate the different kinds of dummy indices and dummy buckets to make the anonymous and the normal histogram differentially private.

G Shuffle Protocol

Protocol $\mathit{\Pi}_{\mathsf{Shuffle}}$

Common input: Sequence of ciphertext pairs $(\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i)_{i\in[n]}$.

1. Shuffle 1, S_1 does:
   (a) Sample a permutation $\sigma_1 \in_R \mathsf{Perms}([n])$.
   (b) Sample randomness $r_i \leftarrow \mathcal{D}_q$ and $s_i \in_R \mathbb{F}_q$ for $i \in [n]$.
   (c) Set $\mathsf{clct}\text{-}u^{(1)}_{\sigma_1(i)} := \mathsf{Randomize}^{\mathsf{cl}}(\mathsf{clct}\text{-}u_i; r_i)$ and $\mathsf{poct}\text{-}v^{(1)}_{\sigma_1(i)} := \mathsf{Randomize}^{\mathsf{po}}(\mathsf{poct}\text{-}v_i; s_i)$ for $i \in [n]$.
   (d) Compute a proof
   $$\pi^{(1)} \leftarrow \mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{Shuffle}}.\mathsf{Prove}\big(((\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i)_{i\in[n]}, (\mathsf{clct}\text{-}u_{\sigma_1(i)}, \mathsf{poct}\text{-}v_{\sigma_1(i)})_{i\in[n]}), (\sigma, (r_i)_{i\in[n]}, (s_i)_{i\in[n]})\big).$$
   (e) Send $(\mathsf{clct}\text{-}u^{(1)}_i, \mathsf{poct}\text{-}v^{(1)}_i)_{i\in[n]}$ with $\pi^{(1)}$ to S_2.
   (f) S_2 verifies the proof and aborts on failure:
   $$\mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{Shuffle}}.\mathsf{Verify}\big(\pi^{(1)}, ((\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i)_{i\in[n]}, (\mathsf{clct}\text{-}u_{\sigma_1(i)}, \mathsf{poct}\text{-}v_{\sigma_1(i)})_{i\in[n]})\big) \overset{?}{=} \top.$$
2. Shuffle 2: S_2 and S_1 perform the same steps in reverse to obtain $(\mathsf{clct}\text{-}u^{(2)}_i, \mathsf{poct}\text{-}v^{(2)}_i)_{i\in[n]}$.

Common output: Sequence of ciphertext pairs $(\mathsf{clct}\text{-}u^{(2)}_i, \mathsf{poct}\text{-}v^{(2)}_i)_{i\in[n]}$.

Fig. 14: Shuffle and rerandomization protocol for CL/ElGamal ciphertext pairs

H Simulator $\mathcal{S}_{\mathsf{Hist}}$ for the Proof of Π_{Hist}

Initially, the simulator $\mathcal{S}_{\mathsf{Hist}}$ receives corruption messages from the environment $\mathcal{Z}$: at most one of $(\mathsf{Corrupt}, \mathsf{S}_1)$ and $(\mathsf{Corrupt}, \mathsf{S}_2)$, and at most $N-1$ of $(\mathsf{Corrupt}, \mathsf{C}_i)$ for $i \in [N]$. In every case, $\mathcal{S}_{\mathsf{Hist}}$ forwards these messages to $\mathcal{F}_{\mathsf{Hist}}$. Moreover, $\mathcal{S}_{\mathsf{Hist}}$ creates virtual instances of $\mathcal{F}_{\mathsf{Enc}}$ and all parties in its head, and gives $\mathcal{Z}$ control over those parties that correspond to the corrupted parties. Depending on which of the servers is corrupted, we distinguish the following three simulation strategies.

Simulator $\mathcal{S}_{\mathsf{Hist}}$ for Π_{Hist} (Case: One malicious server)

Let c and h be the indices of the corrupted and honest server, respectively.

Init Simulate the key generation with $\mathcal{F}_{\mathsf{Enc}}$, store the key pairs $(\mathsf{clsk}, \mathsf{clpk}), (\mathsf{posk}, \mathsf{popk})$.

Client Input – Simulating honest parties' inputs:
After receiving $(\mathsf{InputReceived}, \mathsf{C}_i)$ for some uncorrupted C_i from $\mathcal{F}_{\mathsf{Hist}}$, run the Client Input protocol for the simulated C_i with inputs $u_i = 0$ and $v_i = 0$.
- Extracting corrupt parties' inputs:
On receiving message $(\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i, \pi_{C,i})$ from a corrupted C_i to S_h, verify the proof. If the input is valid, decrypt $u_i \leftarrow \mathsf{Dec}^{\mathsf{cl}}(\mathsf{clsk}, \mathsf{clct}\text{-}u_i)$ and $v_i \leftarrow \mathsf{Dec}^{\mathsf{po}}(\mathsf{posk}, \mathsf{poct}\text{-}v_i)$, and send $(\mathsf{Input}, u_i, v_i)$ on behalf of C_i to $\mathcal{F}_{\mathsf{Hist}}$. Otherwise, send $(\mathsf{Input}, \bot, \bot)$.
- Keep track of the set $I \subseteq [N]$ of indices of clients that provided input.

Eval 1. Agree on client contributions:
   (a) Set $(\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i) := (\bot, \bot)$ for all $i \in [N] \setminus I$ (Step 1a of Π_{Hist}).
   (b) Send the ciphertexts $[\![(\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i)]\!]_{i\in[N]}$ to S_c (Step 1b of Π_{Hist}).
   (c) Receive ciphertexts $[\![(\overline{\mathsf{clct}\text{-}u_i}, \overline{\mathsf{poct}\text{-}v_i})]\!]_{i\in[N]}$ from S_c, and compute index set $J \subseteq I$ as in Equation (17) corresponding to the proofs π_i received by S_h (Step 1c of Π_{Hist}).
   (d) Send $(\mathsf{Influence\text{-}1}, J)$ to $\mathcal{F}_{\mathsf{Hist}}$.
   (e) Filter the valid contributions to obtain a list M' of length $|J|$ (Step 1d of Π_{Hist}).
2. Add dummy contributions by simulating Step 2 as in Π_{Hist}.
   (a) Simulate the call to $\mathsf{PO\text{-}Zero}$ of $\mathcal{F}_{\mathsf{Enc}}$ and generate $\mathsf{poct}\text{-}0$.
   (b) Simulate frequency dummies:
      i. Let $\mathsf{id}_1, \mathsf{id}_2$ be two fresh IDs. Simulate $\mathcal{F}_{\mathsf{Com}}$ to send $(\mathsf{Committed}, \mathsf{id}_\mathsf{h})$ to S_c. Receive $(\mathsf{Commit}, \mathsf{id}_\mathsf{c}, D_{IF}^{(\mathsf{c})}, P_{IF}^{(\mathsf{c})})$ from S_c.
      ii. Verify the ciphertexts in $D_{IF}^{(\mathsf{c})}$ with the proofs in $P^{(\mathsf{c})}$ (as in Step 2(b)iv of Π_{Hist}). If successful, extract S_c's dummy indices by decrypting: $\overline{D_{IF}}^{(\mathsf{c})} \leftarrow [\![\mathsf{Dec}^{\mathsf{cl}}(\mathsf{clsk}, \mathsf{clct}\text{-}d) \mid \mathsf{clct}\text{-}d \in D_{IF}^{(\mathsf{c})}]\!]$. Otherwise, set $\overline{D_{IF}}^{(\mathsf{c})} := [\![]\!]$. Send $(\mathsf{Influence\text{-}2}, \overline{D_{IF}}^{(\mathsf{c})})$ to $\mathcal{F}_{\mathsf{Hist}}$.
      iii. Receive $(\mathsf{Leakage\text{-}1}, n_{DIF}^{(\mathsf{h})})$ from $\mathcal{F}_{\mathsf{Hist}}$. Create a list of ciphertexts $D_{IF}^{(\mathsf{h})} \leftarrow [\![\mathsf{Enc}^{\mathsf{cl}}(\mathsf{clpk}, 0)]\!]_{i\in[n_{DIF}^{(\mathsf{h})}]}$ with a corresponding list $P_{IF}^{(\mathsf{h})}$ of proofs according to $\mathsf{R}_{\mathsf{CL\text{-}PoPK}}$ of matching cardinality.
      iv. Send $(\mathsf{Opened}, \mathsf{id}_\mathsf{h}, D_{IF}^{(\mathsf{h})}, P_{IF}^{(\mathsf{h})})$ to S_c.
      v. Receive $(\mathsf{Open}, \mathsf{id}_\mathsf{c})$ from S_c. If the proofs did not verify in Step 2(b)ii above, simulate an abort.
   (c) Simulate the shuffling of M', $D_{IF}^{(1)}$, and $D_{IF}^{(2)}$ (Step 2c of Π_{Hist}) according to Π_{Shuffle} to obtain M''. Let $N'' := |M''|$.
   (d) Simulate duplication dummies:
      i. Receive $(\mathsf{Leakage\text{-}2}, n_{DIP})$ from $\mathcal{F}_{\mathsf{Hist}}$.
      ii. Simulate the (Rand, N'', p) call to $\mathcal{F}_{\mathsf{Rand}}$ to output a string $\mathsf{rnd} \in \{0,1\}^{N''}$ with Hamming weight n_{DIP}.
      iii. Compute D_{ID} as in Step 2(d)ii of Π_{Hist}.
   (e) Simulate the shuffling of M'' and D_{ID} (Step 2e of Π_{Hist}) according to Π_{Shuffle} to obtain M'''. Let $N''' := |M'''|$.
3. Receive $(\mathsf{Leakage\text{-}2}, \mathcal{H}^A)$ from $\mathcal{F}_{\mathsf{Hist}}$, and set $B := |\mathcal{H}^A|$. Simulate the OPRF evaluation protocol Π_{OPRF} as described in Figure 16.
4. Aggregation: Simulate Step 4 as in Π_{Hist}.

5. Add dummy buckets
   (a) Simulate the call to $(\mathsf{CL\text{-}Constant}, u_D)$ of $\mathcal{F}_{\mathsf{Enc}}$ and generate $\mathsf{poct}\text{-}u_D$.
   (b) Receive $(\mathsf{Leakage\text{-}3}, n_{DB}^{(\mathsf{h})})$ from $\mathcal{F}_{\mathsf{Hist}}$.
   (c) Create a list of ciphertexts $D_B^{(\mathsf{h})} \leftarrow [\![\mathsf{Enc}^{\mathsf{po}}(\mathsf{popk}, 0)]\!]_{i \in [n_{DB}^{(\mathsf{h})}]}$ with a corresponding list $P_B^{(\mathsf{h})}$ of proofs according to $\mathsf{R}_{\mathsf{PO\text{-}PoPK}}$ of matching cardinality, and send $D_B^{(\mathsf{h})}, P_B^{(\mathsf{h})}$ to $\mathsf{S_c}$.
   (d) Receive $D_B^{(\mathsf{c})}, P_B^{(\mathsf{c})}$ from $\mathsf{S_c}$. Verify the ciphertexts in $D_I^{(\mathsf{c})}$ with the proofs in $P_B^{(\mathsf{c})}$ (as in Step 5d of Π_{Hist}), and simulate abort not successful.
   (e) Extract $\mathsf{S_c}$'s dummy buckets by decrypting: $\overline{D_B}^{(\mathsf{c})} \leftarrow [\![\mathsf{Dec}^{\mathsf{po}}(\mathsf{posk}, \mathsf{poct}\text{-}d) \mid \mathsf{poct}\text{-}d \in D_B^{(\mathsf{c})}]\!]$, and send $(\mathsf{Influence\text{-}3}, \overline{D_B}^{(\mathsf{c})})$ to $\mathcal{F}_{\mathsf{Enc}}$.
   (f) Define H' and B' as in Step 5e of Π_{Hist}.
6. Simulate the shuffle in Step 6 as in Π_{Hist} to obtain $(\mathsf{clct}\text{-}\bar{u}_i, \mathsf{poct}\text{-}\bar{v}_i)_{i \in [B']}$.
7. Add noise to outputs
   (a) Simulate Step 7 for $\mathsf{S_h}$ and simulate the encrypted noise by creating ciphertexts $\mathsf{poct}\text{-}\zeta_i^{(\mathsf{h})} \leftarrow \mathsf{Enc}^{\mathsf{po}}(\mathsf{popk}, 0)$ for $i \in [B']$.
   (b) Receive $(\mathsf{poct}\text{-}\zeta_i^{(\mathsf{c})}, \pi_{\zeta,i}^{(\mathsf{c})})_{i \in [B']}$ from $\mathsf{S_c}$. Verify the ciphertexts $\mathsf{poct}\text{-}\zeta_i^{(\mathsf{c})}$ with the proofs $\pi_{\zeta,i}^{(\mathsf{c})}$ for $i \in [B']$ (as in Step 7c of Π_{Hist}) , and simulate abort not successful.
   (c) Decrypt $\zeta_i^{(\mathsf{c})} \leftarrow \mathsf{Dec}^{\mathsf{po}}(\mathsf{posk}, \mathsf{poct}\text{-}\zeta_i^{(\mathsf{c})})$ for $i \in [B']$.
   (d) Send $(\mathsf{Influence\text{-}4}, (\zeta_i^{(\mathsf{c})})_{i \in [B']})$ to $\mathcal{F}_{\mathsf{Hist}}$.
   (e) Compute the ciphertexts $\mathsf{poct}\text{-}\tilde{v}_i \leftarrow \mathsf{poct}\text{-}\bar{v}_i + \mathsf{poct}\text{-}\zeta_i^{(\mathsf{c})} + \mathsf{poct}\text{-}\zeta_i^{(\mathsf{h})}$ for $i \in [B']$.
8. Thresholding
   (a) Receive $(\mathsf{Leakage\text{-}4}, V^{\perp}, \mathcal{H}_I^P)$ from $\mathcal{F}_{\mathsf{Hist}}$, which are of the form $\mathcal{H}_I^P = [\![(i, \bar{u}_i, \tilde{v}_i)]\!]_{i \in I}$ $V^{\perp} = [\![(i, \tilde{v}_i)]\!]_{i \in [B'] \setminus I}$ for some $I \subseteq [B']$.
   (b) Simulate the calls to $(\mathsf{PO\text{-}Decrypt}, \mathsf{poct}\text{-}\tilde{v}_i)$ of $\mathcal{F}_{\mathsf{Enc}}$ by returning $\tilde{v}_i$ (and leaking $g^{\tilde{v}_i}$ to $\mathcal{Z}$) for $i \in [B']$.
   (c) Simulate the calls to $(\mathsf{CL\text{-}Decrypt}, \mathsf{poct}\text{-}\bar{u}_i)$ of $\mathcal{F}_{\mathsf{Enc}}$ by returning $\bar{u}_i$ for $i \in I$.
9. Send $(\mathsf{continue})$ to $\mathcal{F}_{\mathsf{Hist}}$.

Fig. 15: Simulator $\mathcal{S}_{\mathsf{Hist}}$ for the security proof Π_{Hist} for Theorem 5 in the case of one corrupted server.

Simulation subprocedure for the Π_{OPRF} subprotocol

This procedure is part of the simulation of Eval in Π_{Hist} and is executed in Step 3 of Figure 15. Recall that we want to simulate the execution of the Π_{OPRF} subprotocol on a list of CL ciphertexts $[\![\mathsf{clct}\text{-}u_i']\!]_{i \in [N''']}$. Moreover, we also know the corresponding secret key clsk^a and the anonymous histogram $\mathcal{H}^A$.

Common steps
1. Key generation: simulate the call to $\mathsf{CL\text{-}Random}$ of $\mathcal{F}_{\mathsf{Enc}}$ by sampling a key $k \in_R \mathbb{F}_q$ and outputting $\mathsf{clct}\text{-}k \leftarrow \mathsf{Enc}^{\mathsf{cl}}(\mathsf{clpk}, k)$.

Case of corrupted S_1
2. Preparation:
   (a) Simulate according to the protocol (Step 2 of Π_{OPRF}).
   (b) Additionally decrypt $a_i \leftarrow \mathsf{Dec}^{\mathsf{cl}}(\mathsf{clsk}, \mathsf{clct}\text{-}a_i)$ for each $i \in [N''']$. (If $a_i = 0$ we abort, but this happens with negligible probability.)
3. Exponentiation:
   (a) Set $h_i := g^{\frac{1}{a_i}}$ for $i \in [N''']$.
   (b) Simulate the call to $\mathsf{DecExpInv}$ of $\mathcal{F}_{\mathsf{Enc}}$ by outputting $[\![h_1, \ldots, h_{N'''}]\!]$ and leaking $[\![a_1, \ldots, a_{N'''}]\!]$ to $\mathcal{Z}$.
4. Completion:
   (a) Let $\mathcal{T} := [\![]\!]$. For each $n \in \mathcal{H}^A$, sample $t^{(n)} \in_R \mathbb{G}$ and add n copies of it to $\mathcal{T}$.

(b) Let $[\![t_1, \ldots, t_{N'''}]\!] := \mathrm{Shuffle}(\mathcal{T})$. of the elements of $\mathcal{T}$.

(c) Simulate the proofs $\pi_i^{\mathsf{oprf2}} \leftarrow \mathit{\Pi}^{\mathsf{ZK}}_{\mathsf{ExpCom}}.\mathsf{SimProve}\big((h_i, t_i, \mathsf{cm}_i)\big)$. This works, because the commitment scheme is perfectly hiding, so there exists a valid witness.

Case of corrupted S_2

2. Preparation:
   (a) Receive $(\mathsf{clct}\text{-}a_i, \mathsf{cm}_i, \pi_i^{\mathsf{oprf1}})$ for each $i \in [N''']$ from S_2.
   (b) Simulate abort if the proofs are not valid.
3. Exponentiation
   (a) Decrypt $a_i \leftarrow \mathsf{Dec}^{\mathsf{cl}}(\mathsf{clsk}, \mathsf{clct}\text{-}a_i)$ and $u'_i \leftarrow \mathsf{Dec}^{\mathsf{cl}}(\mathsf{clsk}, \mathsf{clct}\text{-}u'_i)$, and compute $r_i := a_i \cdot (u'_i + k)^{-1}$ for all $i \in [N''']$.
   (b) Sample $t_1, \ldots, t_{N'''} \in \mathbb{G}$ as in Step 4 in the case of corrupted S_1.
   (c) Set $h_i := t_i^{\frac{1}{r_i}}$.
   (d) Simulate the call to $\mathsf{DecExpInv}$ of $\mathcal{F}_{\mathsf{Enc}}$, by outputting $(h_1, \ldots, h_{N'''})$.
4. Completion:
   (a) Receive $t'_i \in \mathbb{G}$ and π_i^{oprf2} from S_2 (it should be $t_i = t'_i$).
   (b) Simulate abort if the proofs do not verify.

[a] NB: While we can use clsk to decrypt in the UC simulation, we will use rewinding and extractions from proofs of knowledge when proving indistinguishability of the simulation based on the security of the encryption scheme.

Fig. 16: Subprocedure that is called in Step 3 of the simulator $\mathcal{S}_{\mathsf{Hist}}$ (Figure 15) for the security proof $\mathit{\Pi}_{\mathsf{Hist}}$ for Theorem 5.

Simulator $\mathcal{S}_{\mathsf{Hist}}$ for $\mathit{\Pi}_{\mathsf{Hist}}$ (Case: Honest servers)

Init Simulate the key generation with $\mathcal{F}_{\mathsf{Enc}}$, store the key pairs $(\mathsf{clsk}, \mathsf{clpk}), (\mathsf{posk}, \mathsf{popk})$.
Client Input On receiving message $(\mathsf{clct}\text{-}u_i, \mathsf{poct}\text{-}v_i, \pi_i)$ and $(\mathsf{clct}\text{-}u_i{}', \mathsf{poct}\text{-}v_i{}', \pi'_i)$ from a corrupted C_i to S_1 and S_2, respectively, verify the proofs. If they verify and additionally $\mathsf{clct}\text{-}u_i = \mathsf{clct}\text{-}u_i{}'$ and $\mathsf{poct}\text{-}v_i = \mathsf{poct}\text{-}v_i{}'$, then decrypt $u_i \leftarrow \mathsf{Dec}^{\mathsf{cl}}(\mathsf{clsk}, \mathsf{clct}\text{-}u_i)$ and $v_i \leftarrow \mathsf{Dec}^{\mathsf{po}}(\mathsf{posk}, \mathsf{poct}\text{-}v_i)$. Send $(\mathsf{Input}, u_i, v_i)$ on behalf of C_i to $\mathcal{F}_{\mathsf{Hist}}$. Otherwise, send $(\mathsf{Input}, \perp)$.
Eval Nothing to be done, since the clients to not participate in the evaluation.

Fig. 17: Simulator $\mathcal{S}_{\mathsf{Hist}}$ for the security proof $\mathit{\Pi}_{\mathsf{Hist}}$ for Theorem 5 in the case no corrupted server.

I Missing Proofs of Claims in the Proof of Theorem 5

I.1 Proof of Claim 1 in Theorem 5

Proof of Claim 1. We prove the indistinguishability based on the IND-CPA security of the modified HSM-CL encryption scheme (Theorem 6).

Suppose towards contradiction that there exists an environment $\mathcal{Z}$ that can distinguish between $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{real}}(\lambda)$ and $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid}\text{-}1}(\lambda)$ with non-negligible advantage $\mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},1}(\lambda)$. We now use $\mathcal{Z}$ to construct a distinguisher $\mathcal{D}$ for the HSM-CL IND-CPA game $\mathsf{Game}^{\mathsf{IND\text{-}CPA,cl}}_{\mathcal{D},b}$ with $b \in_R \{0,1\}$.

In the IND-CPA game, $\mathcal{D}$ receives an HSM-CL public key clpk. We setup an execution with $\mathcal{Z}$ that is identical to $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid}\text{-}1}(\lambda)$ until we let $\mathcal{F}_{\mathsf{Enc}}$, return the given clpk when (KeyGen) is called instead of generating a fresh HSM-CL key pair.

Note that in the reduction, we do not know the secret key clsk. Hence, we cannot use it to decrypt the inputs u_i to Π_{OPRF}. Instead we make use that each party provides proofs of plaintext knowledge ($\Pi^{\mathsf{ZK}}_{\mathsf{CL\text{-}PoPK}}$) for their inputs. So we use rewinding of the execution to extract all plaintexts (this is not the UC simulation, so it does not need to be straight-line), and we keep track of which plaintext is encrypted in which ciphertext. For this to succeed, we also use that we can extract the used permutations from the shuffle proofs $\Pi^{\mathsf{ZK}}_{\mathsf{Shuffle}}$.

We continue the execution normally, until $(\mathsf{CL\text{-}Random})$ of $\mathcal{F}_{\mathsf{Enc}}$ is called in Step 1. Now we sample $k_0, k_1 \in_R \mathbb{F}_q$ and submit them to the IND-CPA challenger. We obtain a ciphertext $\mathsf{clct}\text{-}k_b \leftarrow \mathsf{Enc}^{\mathsf{cl}}(\mathsf{clpk}, k_b)$ which we let $\mathcal{F}_{\mathsf{Enc}}$ return as $\mathsf{clct}\text{-}k := \mathsf{clct}\text{-}k_b$ for $(\mathsf{CL\text{-}Random})$. We continue as in $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid}\text{-}1}(\lambda)$, except that we use $k' := k_0$ when computing the F^{DY} outputs. The output of $\mathcal{D}$ is whatever $\mathcal{Z}$ outputs.

If $b = 1$, then the view of $\mathcal{Z}$ is identical to its view in $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid}\text{-}1}(\lambda)$ because the ciphertext $\mathsf{clct}\text{-}k$ contains a random field element, and the F^{DY} evaluation uses an independent random key k'. Otherwise, if $b = 0$, $\mathcal{Z}$'s view is identical to its view in $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{real}}(\lambda)$, since then the F^{DY} evaluation uses the same key k that is contained in $\mathsf{clct}\text{-}k$.

Hence, $\mathcal{D}$ would win the game $\mathsf{Game}^{\mathsf{IND\text{-}CPA,cl}}_{\mathcal{D},b}(\lambda)$ with non-negligible advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA,cl}}_{\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},1}(\lambda)$, and we reached a contradiction. Therefore such an environment $\mathcal{Z}$ cannot exist. ∎

I.2 Proof of Claim 2 in Theorem 5

Proof of Claim 2. We prove the indistinguishability based on the ℓ-non-adaptive PRF security of the Dodis-Yampolskiy PRF (Theorem 1), where we set $\ell := N'''$

Suppose towards contradiction that there exists an environment $\mathcal{Z}$ that can distinguish between $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid}\text{-}1}(\lambda)$ and $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid}\text{-}2}(\lambda)$ with non-negligible advantage $\mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},2}(\lambda)$. We now use $\mathcal{Z}$ to construct a distinguisher $\mathcal{D}$ for the ℓ-non-adaptive PRF security game $\mathsf{Game}^{\ell\text{-}\mathsf{naPRF}}_{\mathcal{D},b}(\lambda)$ with $b \in_R \{\mathsf{real}, \mathsf{rand}\}$ (Definition 4).

In the security game $\mathsf{Game}^{\ell\text{-}\mathsf{naPRF}}_{\mathcal{D},b}(\lambda)$, $\mathcal{D}$ first specifies ℓ inputs $x_1, \ldots, x_\ell \in \mathbb{F}_q$, and then obtains either outputs $y_i \in \mathbb{G}$ of a random oracle ($y_i = H(x_i)$ if $b = \mathsf{rand}$) or outputs of the DY-PRF ($y_i = \mathsf{F}^{\mathsf{DY}}(k', x_i)$ if $b = \mathsf{real}$) for a random $k' \in_R \mathbb{F}_q$.

We setup an execution with $\mathcal{Z}$ that is identical to $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid}\text{-}1}(\lambda)$ and $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid}\text{-}2}(\lambda)$ until the beginning of Π_{OPRF}. At this point we decrypt the inputs $u_i \leftarrow \mathsf{Dec}^{\mathsf{cl}}(\mathsf{clsk}, \mathsf{clct}\text{-}u_i)$ for $i \in [\ell]$. We submit them to the $\mathsf{Game}^{\ell\text{-}\mathsf{naPRF}}_{\mathcal{D},b}$ challenger and obtain $y_1, \ldots, y_\ell \in g$. The remainder of the execution is identical to the hybrid games, except that we inject the values y_i instead of $\mathsf{F}^{\mathsf{DY}}(k', u_i)$ and $H(u_i)$, respectively. We let the output of $\mathcal{D}$ be whatever $\mathcal{Z}$ outputs.

Consequently, if $b = \mathsf{real}$, then the view of $\mathcal{Z}$ is identical to its view in $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid}\text{-}1}(\lambda)$, and if $b = \mathsf{rand}$, it is identical to its view in $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid}\text{-}2}(\lambda)$. Hence, $\mathcal{D}$ would win the game $\mathsf{Game}^{\ell\text{-}\mathsf{naPRF}}_{\mathcal{D},b}(\lambda)$ with non-negligible advantage $\mathsf{Adv}^{\ell\text{-}\mathsf{naPRF}}_{\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},2}(\lambda)$, and we reached a contradiction. Therefore such an environment $\mathcal{Z}$ cannot exist. ∎

I.3 Proof of Claim 3 in Theorem 5

Proof of Claim 3. For this indistinguishability proof, we again us the security of the encryption schemes. However, we do not use IND-CPA security directly, but the property that both encryption schemes admit lossy public keys (Theorems 6 and 7).

By a hybrid argument, we can show that indistinguishability also holds for pairs of public keys that are both lossy or both working:

$$\begin{aligned}\{(\mathsf{KeyGen}^{\mathsf{cl}}_0(), \mathsf{KeyGen}^{\mathsf{po}}_0())\} &\approx_C \{(\mathsf{KeyGen}^{\mathsf{cl}}_1(), \mathsf{KeyGen}^{\mathsf{po}}_0())\} \\ &\approx_C \{(\mathsf{KeyGen}^{\mathsf{cl}}_1(), \mathsf{KeyGen}^{\mathsf{po}}_1())\}.\end{aligned} \tag{28}$$

Let $\mathsf{Game}^{\mathsf{IND\text{-}LK}}_{\mathcal{D},b}$ with $b \in_R \{0,1\}$ denote the security game that gives a distinguisher $\mathcal{D}$ a sample of the distribution $(\mathsf{clpk}, \mathsf{popk}) \leftarrow (\mathsf{KeyGen}^{\mathsf{cl}}_b(), \mathsf{KeyGen}^{\mathsf{po}}_b())$.

We follow the strategy of the proof of Π^q_{ABB} of [BDO23]: Suppose towards contradiction that there exists an environment $\mathcal{Z}$ that can distinguish between $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid\text{-}2}}(\lambda)$ and $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{ideal}}(\lambda)$ with non-negligible advantage $\mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},3}(\lambda)$. We now use $\mathcal{Z}$ to construct a distinguisher $\mathcal{D}$ for $\mathsf{Game}^{\mathsf{IND\text{-}LK}}_{\mathcal{D},b}$.

First, we sample a bit $b^*_{\mathcal{D}} \in_R \{0,1\}$. Depending on this we execute either $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{ideal}}(\lambda)$ (if $b^*_{\mathcal{D}} = 0$) or $\mathsf{Game}^{\mathsf{Hist}}_{\mathcal{Z},\mathsf{hybrid\text{-}2}}(\lambda)$ (if $b^*_{\mathcal{D}} = 1$), with two differences:

1. First, we now simulate all the zero-knowledge proofs, even if we knew the corresponding witnesses.
2. Second, since we construct a distinguisher for the $\mathsf{Game}^{\mathsf{IND\text{-}LK}}_{\mathcal{D},b}$ game, we do not know the secret keys $(\mathsf{clsk}, \mathsf{posk})$ corresponding to the given public keys $(\mathsf{clpk}, \mathsf{popk})$. Therefore, we employ the same strategy as in the proof of Claim 1 and extract all plaintexts from the corresponding proofs of plaintext knowledge.

Let $b^*_{\mathcal{Z}} \in \{0,1\}$ denote the environment's output. If $b^*_{\mathcal{Z}} = b^*_{\mathcal{D}}$, then $\mathcal{D}$ outputs $b' := 1$, and if $b^*_{\mathcal{Z}} \neq b^*_{\mathcal{D}}$, $\mathcal{D}$ outputs $b' := 0$. We distinguish two cases:

– Case $b = 0$: All encryptions are made with lossy public keys so that all ciphertexts that the environment $\mathcal{Z}$ sees are statistically indistinguishable from encryptions of zero. Since additionally all the proofs are simulated, the view of $\mathcal{Z}$ in the hybrid-2 execution is statistically independent of the honest parties' inputs and statistically coincides with its view in the ideal execution. So we have

$$\Pr\left[\mathcal{D} \text{ outputs } 1 \mid b = 0\right] = 1/2 - \mathsf{negl}(\sigma).$$

– Case $b = 1$: Since the public keys are working, the execution is statistically indistinguishable (the only difference is the simulation of the zero-knowledge proofs) to the ideal execution (if $b^*_{\mathcal{D}} = 0$) or the hybrid-2 execution (if $b^*_{\mathcal{D}} = 1$). Hence, we have

$$\Pr\left[\mathcal{D} \text{ outputs } 1 \mid b = 1\right] = 1/2 + \mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},3}(\lambda) - \mathsf{negl}(\sigma).$$

Combining both case, we obtain that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IND\text{-}LK}}_{\mathcal{D}}(\lambda) &= \left|\Pr\left[\mathsf{Game}^{\mathsf{IND\text{-}LK}}_{\mathcal{D},1}(\lambda) = 1\right] - \Pr\left[\mathsf{Game}^{\mathsf{IND\text{-}LK}}_{\mathcal{D},0}(\lambda) = 1\right]\right| \\ &= \mathsf{Adv}^{\mathsf{Hist}}_{\mathcal{Z},3}(\lambda)/2 - \mathsf{negl}(\sigma).\end{aligned}$$

If σ is chosen such that $\mathsf{negl}(\sigma)$ is also negligible in λ, then we have constructed a distinguisher $\mathcal{D}$ that has non-negligible advantage in the $\mathsf{Game}^{\mathsf{IND\text{-}LK}}_{\mathcal{D},b}(\lambda)$ game. We reached a contradiction, and therefore such an environment $\mathcal{Z}$ cannot exist. ■