Verifiable Distributed Aggregation Functions

Hannah Davis
University of California, San Diego

Christopher Patton
Cloudflare

Mike Rosulek
Oregon State University

Phillipp Schoppmann
Google

2023/07/12

Abstract

The modern Internet is built on systems that incentivize collection of information about users. In order to minimize privacy loss, it is desirable to prevent these systems from collecting more information than is required for the application. The promise of *multi-party computation* is that data can be aggregated without revealing individual measurements to the data collector. This work offers a provable security treatment for "Verifiable Distributed Aggregation Functions (VDAFs)", a class of multi-party computation protocols being considered for standardization by the IETF.

We propose a formal framework for the analysis of VDAFs and apply it to two constructions. The first is Prio3, one of the candidates for standardization. This VDAF is based on the Prio system of Corrigan-Gibbs and Boneh (NSDI 2017). We prove that Prio3 achieves our security goals with only minor changes to the draft. The second construction, called Doplar, is introduced by this paper. Doplar is a round-reduced variant of the Poplar system of Boneh et al. (IEEE S&P 2021), itself a candidate for standardization. The cost of this improvement is a modest increase in overall bandwidth and computation.

Change Log

The proceedings version of this paper appears at PETS 2023. This is the full version.

- 2023/06/15: Proceedings version submitted. This version includes minor changes to address the remaining feedback from the PETS 2023 program committee.
- 2023/02/22: Fix a minor bug in the Doplar spec (Section 5). Previously we had included the IDPF tree level in the derivation of the joint randomness parts; but these are shared across all levels. Remove the level from the joint randomness parts binder; add the level to the joint randomness seed binder; and propagate these changes through the proofs of robustness and privacy. (No changes to the reductions or bounds were required.)
- 2023/02/03: Initial ePrint submission.

1 Introduction

Operating a complex software system, such as an operating system, web browser, or web service, often requires measuring the behavior of the system's users. When used for a specific purpose, such measurements are often only consumed in some aggregated form, e.g., $F(m_1, \ldots, m_{ct})$ for some specific function F, rather than the individual measurements $m_1, \ldots, m_{ct}$. But in conventional systems, the measurements are revealed to the operator as a matter of course, resulting in an increased capability to surveil users. Consider the following motivating examples:

1. *Identifying misbehaving or malicious origins.* To detect bugs or attack vectors, a browser vendor might want to know how often establishing a connection to a given origin or loading a given web page triggers a specific event [48]. But logging these events and aggregating them in the clear risks exposing browser history.

2. *Measuring ad conversion rates.* Today advertising is a significant revenue source for many web service providers. In order to accurately assess the value of an ad campaign, the service provider and advertiser might want to measure how many people who clicked on a given ad made a purchase [2].

3. *Classifying malicious client behavior.* Many operators benefit from the ability to classify (or predict) user behavior automatically, and in real-time. For example, anomaly detection systems use machine learning models, trained and validated on requests from real clients, to classify fraudulent or otherwise malicious behavior [46].

These applications require only aggregates; by collecting individual measurements, the operator learns more information than is ultimately used for the intended purpose. One way out of this predicament is *multi-party computation (MPC)*, which allows computing some function of private inputs distributed across multiple parties, without revealing these private inputs. In this paper, we consider a class of MPC protocols in which the bulk of the computation is outsourced to a small set of non-colluding servers.

Recent attention from the MPC community on problems like these has yielded solutions that are practical enough for real-world deployment [32, 23, 17, 18, 5, 10]. Notable examples include Mozilla's Origin Telemetry project [48] and the COVID-19 Exposure Notification Private Analytics system developed jointly by Apple and Google [7]. The success of these projects spurred the formation of a working group within the Internet Engineering Task Force (IETF) whose objective is to standardize MPC for "Privacy-Preserving Measurement (PPM)" [1], thereby improving interoperability and providing a deployment roadmap for new schemes.

The primary goal of this paper is to lay some of the groundwork for the provable security analysis that will be needed to support this effort. We formalize a syntax and set of security definitions for a particular class of MPC protocols from the literature [23, 17, 18, 5] of interest to the working group. Our definitions unify previous ones into an explicit, game-based framework that accounts for practical matters not attended to in prior work.

We apply our definitional framework to two constructions. The first is a candidate for standardization based on the Prio scheme designed by Corrigan-Gibbs and Boneh [23]; we show that this protocol meets our security goals with only minor changes. Another candidate for standardization is the more recent Poplar scheme due to Boneh et al. [18]; we introduce and analyze a variant of this protocol that has improved round complexity.

Overview. The PPM working group plans to develop multiple protocol standards, one of which is the focus of this work. The *Distributed Aggregation Protocol (DAP)* standard [30] centers around the execution of a particular class of MPC protocols, called *Verifiable Distributed Aggregation Functions (VDAFs)* [9]. A VDAF is used to securely compute some **aggregation function** F over a set of measurements generated by the **clients**. To protect their privacy, the measurements are secret-shared and the computation of the aggregate is distributed amongst multiple, non-colluding aggregation servers (called **aggregators** hereafter). Execution of a VDAF involves four basic steps (illustrated in Figure 1):

- Shard: Each client shards its measurement m_i into **input shares** and sends one share to each aggregator. In this work, we sometimes refer to this sequence of input shares as the client's **report**.

- Prepare: After receiving a report from a client, the aggregators gossip amongst themselves in order to prepare their shares for aggregation. This involves refining the shares into an aggregatable form and verifying that the outputs are "well-formed", e.g., that they correspond to an integer in a given range, or correspond to a one-hot vector (a vector that is non-zero in at most one position). We call the outputs of this process the **refined shares**.

- Aggregate: Once an aggregator has recovered the desired number of refined shares, it combines them into its share of the aggregate result, called an **aggregate share**. It then sends this to the data consumer, known as the **collector**.

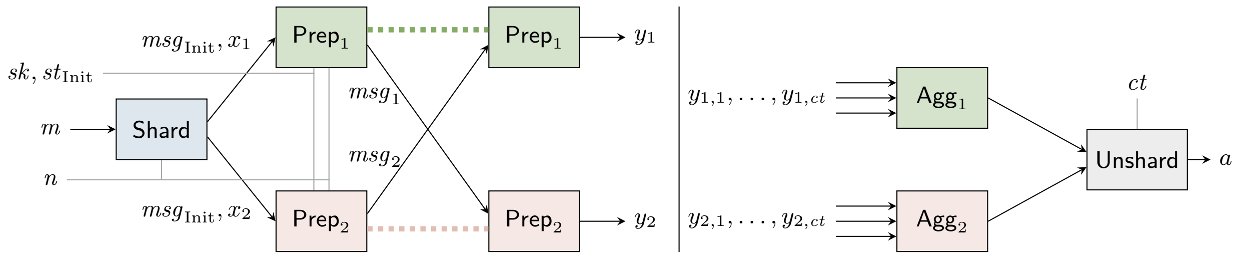

Figure 1: Illustration of (left) sharding and preparation of a single measurement and (right) aggregation and unsharding of a set of measurements. All parameters are defined in Section 3.

- Unshard: Finally, the collector combines each of the aggregate shares into $F(m_1, \ldots, m_{ct})$.

Why standardize VDAFs? The case for standardizing this class of MPC protocols is made by the aforementioned deployments of Prio [48, 7], of which VDAFs are a natural generalization. The key feature that makes these protocols widely applicable and suited for Internet scale is that the expensive part of the computation (Shard/Prepare) is fully parallelizable across all reports being aggregated. This means that deployments can be scaled to such a degree that the time spent on executing the VDAF is primarily *network-bound* rather than *CPU-bound*. It is less clear (at least to those in the PPM working group) whether MPC techniques where the computations depend on all reports (e.g., oblivious sorting [53] or shuffling [6, 10]) would scale in the same way.

This feature also implies that VDAFs are only suitable for aggregation functions F that can be decomposed into f, g for which $F(m_1, \ldots, m_{ct}) = f(g(m_1), \ldots, g(m_{ct}))$, where g may be non-linear, but f must be affine. Indeed, the goal is not to encompass all possible MPC schemes, but a particular, useful, and highly parallelizable class of them. VDAFs can be used for a variety of aggregation tasks, including: simple statistics like sum, mean, standard deviation, quantile estimates, or linear regression [23]; a step of a gradient descent [37]; or heavy hitters (see below).

Security goals. The PPM working group's primary goal for VDAFs (cf. [30, Section 7]) is that they are **private** in the sense that the attacker learns nothing about the measurements $m_1, \ldots, m_{ct}$ beyond what it can infer from the aggregate result $F(m_1, \ldots, m_{ct})$. An active attacker who corrupts the collector and a fraction of the aggregators (typically all but one) and controls transmission of all messages in the protocol—except, of course, the input shares delivered to honest aggregators. Its corruptions are "static": the set of corrupt parties does not change over the course of the attack.

Another security consideration for VDAFs is that they are **robust** in the sense that the attacker cannot force the collector to compute anything other than the aggregate of honestly generated reports. Here the attacker is a set of malicious clients attempting to corrupt the aggregate result by sending malformed reports. For robustness we assume all of the aggregators execute the protocol correctly. Otherwise, a corrupt aggregator could trivially corrupt the result by sending the collector a malformed aggregate share.

We formalize these security notions in the game-playing paradigm [14]. First, in Section 3.2 we define privacy via an indistinguishability game $\mathsf{Exp}^{\text{priv}}_{\Pi}(A)$ played by an attacker A against VDAF Π. The attacker interacts with the honest parties (i.e., the clients and uncorrupted aggregators) via a set of oracles. These oracles allow A to mount a kind of "chosen batch attack" in which the honest parties process one of two batches of measurements, and A's goal is to determine which was processed. This is analogous to the simulation-based definition of [23, Definition 1], which asks the the attacker to distinguish the protocol's execution from the view generated by a simulator.

We formalize robustness via a game $\mathsf{Exp}^{\text{robust}}_{\Pi}(A)$ (Section 3.2). Here the attacker A—playing the role of a coalition of malicious clients—is given a single oracle that models the execution of the preparation step of VDAF execution on (invalid) reports. The attacker wins if an aggregator ever accepts an invalid share *or* if the aggregators compute refined shares that, when combined, do not correspond to a valid refined measurement. For natural VDAFs, robustness implies robustness in the sense of [23, Definition 6]: namely, the collector is guaranteed to correctly aggregate measurements uploaded by honest clients.

Note on the simulation paradigm. An alternative approach, and one that is more conventional for MPC, is to formulate security in the Universal Composability (UC) framework [21]. This methodology would begin by specifying the "ideal functionality" for computing an aggregation function such that, for any VDAF that securely realizes this functionality, any suitable notion of either privacy or robustness would follow from the UC composition theorem.

While this methodology is attractive, it creates the following difficulty in our setting. Many applications of VDAFs may be willing to tolerate a loose robustness bound (i.e., a non-negligible probability of accepting an invalid share) if doing so leads to better performance or communication. On the other hand, no application can accept a loose bound for privacy. In order to reason about this tradeoff, it is necessary to obtain explicit, concrete bounds for privacy and robustness *separately.* A theorem in the UC framework yields only a single bound, for the "UC-realizability" of the ideal functionality; applying this result directly would lead to parameter choices that might be more conservative than strictly necessary for the given application.

Another consideration is to make our results accessible to the target audience. Applying the UC framework, and interpreting its results, involves a number of subtleties that, based on our own observations, are often misunderstood when translated to practice.[1] One goal of our definitions is to make as explicit as possible all of the requirements an application like DAP [30] needs to meet in order to use VDAFs securely.

Previous definitions. Our definitions in Section 3 can be seen as a more precise (but not necessarily stronger) formulation of the informal definitions given in the original Prio paper [23, Appendix A]. While the authors mention the possibility of using a unified simulation-based security definition for privacy and robustness, they do not provide one.

For Poplar on the other hand, Boneh et al. [18, Appendix A] provide a simulation-based definition for the end-to-end functionality. In order to capture the fact that a malicious server can influence the output of the protocol, they define a leakage function that allows the attacker to perturb the aggregate result with an arbitrary additive offset. While we believe this captures the robustness attacks that are possible for Poplar, it does not immediately generalize to the broader class of functionalities we consider as VDAFs. Also note that Boneh et al. do not provide any proofs using their security definition. (The proofs they do provide are for definitions that are naturally captured by games, e.g., [18, Appendix D].) Finally, the simulation-based security definition of Poplar only considers a single security parameter, something that would need to be overcome to allow for separate security bounds for privacy and robustness.

Constructions. The starting point for our work is draft-irtf-cfrg-vdaf-03 [9], the current draft of the VDAF specification at the time of writing.

The first scheme described in draft-03, called Prio3, is based on Prio [23], but incorporates performance improvements from Boneh et al. [17] (hereafter BBCG+19). Prio3 can be used to compute a wide variety of aggregation functions due to its use of *Fully Linear Proofs (FLPs).* Briefly, an FLP is a special type of zero-knowledge proof that allows the client's input measurement to be validated by the aggregators (e.g., ensure that it is a number in some pre-determined range) who have only secret shares of the input and proof. The FLP designed by BBCG+19 (see [17, Theorem 4.3]) and adopted by the draft (with minor modifications; see [9, Section 7.3]) is expressed in terms of some arithmetic circuit C that takes in the prover's input x and a random string jr computed jointly by the prover and verifier. Computing this joint randomness, verifying the proof, and evaluating $C(x, jr)$ requires just one round of communication among the aggregators.

In Section 4, we prove Prio3 is both robust (Theorem 1) and private (Theorem 2) under the assumption that the underlying FLP is, respectively, *sound* and *honest-verifier zero-knowledge* as defined by BBCG+19. Our analysis unveiled a few subtle design issues in draft-03 that we address here.

The second scheme in draft-03 is called Poplar1 and is based on the recent Poplar protocol from Boneh et al. [18] (BBCG+21). Poplar is designed to solve the private "heavy hitters" problem in which each client submits an arbitrary bitstring α and the collector wants to compute the set of unique strings that occurred at least T times. The key idea of BBCG+21 is an extension of *distributed point functions (DPFs)* [31], where two aggregators hold a share of a "DPF key" that concisely represents a *point function.* A point function evaluates to 0 on every input, except for the distinguished point α, where the function evaluates to some $\beta \neq 0$. By secret sharing the DPF keys generated by the clients, the aggregators can count *how many* clients submitted a particular candidate string without revealing *which* clients submitted it.

[1] For a recent example, consider the standards for PAKEs ("Password-Authenticated Key Exchange") developed by the CFRG. Most of these standards are based on protocols with analysis in the UC framework. For one protocol [4], one question left open by that analysis was how to securely instantiate the "session identifier", one of the artifacts of the ideal functionality. The current draft offers recommendations for choosing the session identifier, but allows applications to ignore this entirely; a game-playing argument was used to justify this (cf [3, Appendix B]).

Poplar1 makes use of an enriched primitive called an *incremental DPF (IDPF)*. IDPF keys can be queried not only at a given point, but a given *prefix*. That is, an *incremental point function* is one that evaluates to 0 on every input except for the set of strings that are a prefix of α. This new primitive gives rise to an efficient solution to the heavy hitters problem that involves running Poplar1 multiple times over the same set of IDPF keys, where each run begins with a set of candidate prefixes computed from the previous run.

To achieve robustness, Poplar1 uses a two-round multi-party computation in which the aggregators verify that the IDPF outputs are well-formed. That means that, compared to Prio3, the Poplar1 VDAF costs one additional round of communication, per report, during the preparation phase. The additional roundtrip is significant from an operational perspective.

In Section 5 we introduce *Doplar*, our modification to Poplar which achieves a one-round preparation. To achieve this, we combine FLPs and methods from distributed point functions in a novel way. We adopt a point-function verification method from De Castro and Polychroniadou [22]. We also introduce a new flavor of *delayed-input* FLPs, which may be of independent interest.

Related Work. Several works have considered private aggregate statistics, relying either on secret-sharing between non-colluding servers [24, 27, 29, 40, 42, 44], or on anonymization networks [50, 35, 20]. However, these works either do not provide privacy against malicious clients or rely on expensive zero-knowledge proofs.

A protocol for Secure Aggregation (SecAgg) in the single-server setting was presented by Bonawitz et al. [16] and subsequently improved by Bell et al. [12, 11]. While SecAgg can provide security against malicious parties, it relies on multiple rounds of interaction between clients and server.

The VDAF abstraction was designed to encompass the architecture of Prio and Poplar in which the expensive portion of the MPC is fully parallelizable. Another example of a VDAF from the literature is the protocol of Addanki et al. [5], which uses boolean (bit-wise) secret sharing instead of arithmetic circuit to improve communication cost from client to aggregator. However, this comes at a cost of weaker privacy, since their protocol does not protect against malicious servers.

There are also protocols that do not fit neatly into the VDAF framework as specified, but which might be adapted into VDAFs in the future. Masked LARK [37] is a proposal by Microsoft for training machine learning models on private data, using secret-sharing and MPC between a set of aggregators. AdScale [32] presents an aggregation system focused on private ads measurement. While designed for a single aggregation server, their construction appears to be amenable to our multi-server setting.

Other protocols in the literature share the same security goals of VDAFs, but do not have the same streaming architecture. One example is the recent "Oblivious Shuffling" protocol due to Anderson et al. [6], which involves an MPC, assisted by a third-party, for unlinking each report from the client that sent it. The online processing for this procedure intrinsically involves all of the reports being shuffled; for VDAFs, all of the online processing is per-report. Similarly, Bell et al. [10] present a protocol for computing sparse histograms with two aggregators that is more efficient than DPFs for large domains, but reveals differentially private views to the aggregators. Again, the protocol crucially relies on shuffling contributions from multiple users. Vogue [39] is a protocol for computing private heavy hitters using three non-colluding servers. The protocol is secure against malicious servers and clients, but again relies on shuffling. Finally, the STAR protocol [25] uses an anonymizing proxy to ensure the collector only learns "popular" measurements, while any measurement that occurs less than a pre-determined threshold is not revealed to any party.

In recent concurrent work, Mouris et al. [47] present another three-party, honest-majority protocol for computing heavy hitters. Their full protocol relies on a secure comparison protocol that is run after the aggregation phase, and thus doesn't immediately fit our setting. However, we believe their input validation protocol can be adapted to obtain a VDAF for heavy hitters that has similar characteristics as our protocol in Section 5. (Indeed their core primitive, which they also call "Verifiable IDPF", bears a striking resemblence to our own VIDPF abstraction.) Likewise, one could get robustness against malicious aggregators in the honest-majority setting by applying their "duplicate aggregator" technique to our protocols. We leave exploration of how to combine our results to future work.

Full version. This is the proceedings version of our paper. The full version [26] includes proofs of all theorems, a notion of "completeness" for VDAFs, and additional remarks and commentary.

2 Preliminaries

This section describes cryptographic primitives on which our constructions are based. We begin with a bit of non-standard notation.

Notation. Let $[i..j]$ denote the set of integers $\{i, \ldots, j\}$ and write $[i]$ as shorthand for the set $[1..i]$. If $\vec{v}$ is a vector, let $\vec{v}[i]$ denote the i-th element of $\vec{v}$. Let $(x,)$ denote the singleton vector with value x and $()$ the empty vector.

In our pseudocode, all variables that are undeclared implicitly have the value $\perp$. Let $y \leftarrow\!\!\$\, \mathcal{S}$ denote sampling y uniformly from a finite set $\mathcal{S}$; let $y \leftarrow\!\!\$\, A(x)$ denote execution of randomized algorithm A; and let $y \leftarrow A(x; r)$ denote execution of randomized algorithm A with coins r. If X is a random variable with support $\{0, 1\}$ we let $\Pr\left[\, X \,\right]$ denote the probability that $X = 1$.

A table T is a map from unique keys to values; we write $\mathrm{T}[K_1, \ldots]$ to denote the value corresponding to key $K_1, \ldots$. We sometimes write a dot "$\cdot$" in place of one of the elements of the key, e.g., "$\mathrm{T}[K_1, \cdot]$" instead of "$\mathrm{T}[K_1, K_2]$". We use this notation to denote the vector of values in the table that match the key pattern. For example, we write $\mathrm{T}[K_1, \cdot]$ for the vector $(\mathrm{T}[K_1, K_2^1], \ldots, \mathrm{T}[K_1, K_2^n])$ where $(K_1, K_2^1), \ldots, (K_1, K_2^n)$ are all of the keys in the table prefixed by K_1, in lexicographic order.

We measure an adversary's runtime by the time it takes to run its experiment to completion, including evaluating its queries.

Pseudorandom Generators. The VDAF spec [9, Section 6.2] calls for a particular type of object they call "pseudorandom generator (PRG)". Unlike the conventional PRGs, these objects are stateful. A PRG is comprised of the following algorithms:

- $\mathsf{PRG.Init}(\mathit{seed} \in \{0, 1\}^\kappa, \mathit{cntxt} \in \{0, 1\}^*) \to \mathit{st} \in \mathcal{Q}$ takes a seed and context string to the initial PRG state. We call κ the **seed length**.

- $\mathsf{PRG.Next}(\mathit{st} \in \mathcal{Q}, \ell \in \mathbb{N}) \to (\mathit{st}' \in \mathcal{Q}, \mathit{out} \in \{0, 1\}^\ell)$ takes in the current PRG state and outputs a string of the desired length.

We also make use of an algorithm Expand[PRG] that uses the given PRG to map a seed and context string to a vector of integers over the modular ring $\mathbb{Z}_p$ for the desired modulus p. We defer to [9, Section 6.2] for the full definition of Expand[PRG].

In our security proofs, we model PRGs as random oracles [13]. In some cases, such as the distributed point functions (DPFs) in Section 5.1, constructions based on computational assumptions are known to be sufficient. We refer to Guo et al. [33, 34] for an overview of the state-of-the-art PRGs for DPFs and similar constructions.

Fully Linear Proof Systems. We recall the definition of FLP systems from BBCG+19 [17]. (Our formulation differs slightly, as we discuss below.) FLPs allow a prover to prove to a verifier, in zero-knowledge, that a secret-shared value has some property required by the application, e.g., the input is a number in the desired range, is a one-hot vector, etc. (The main construction of BBCG+19 allows the validity condition to be expressed in terms of an arithmetic circuit evaluated over the input, similar to more conventional zero-knowledge proof systems.) They are "fully linear" in the sense that verifying the proof involves computing a strictly linear function over both the input and proof. This allows verification to be performed on secret-shared data, leveraging its additive homomorphism property. (This is contrast to prior work on "linear PCPs" [8, 15, 38] in which the verifier has linear access to the proof, but arbitrary access to the input.)

An FLP with finite field $\mathbb{F}$, proof length m, verifier length v, prover randomness length pl, joint randomness length jl, and query randomness length ql is a triple of algorithms FLP defined as follows:

- $\mathsf{FLP.Prove}(x \in \mathbb{F}^n, \mathit{jr} \in \mathbb{F}^{jl}) \to \pi \in \mathbb{F}^m$ is the randomized **proof-generation** algorithm that takes in an input x and joint randomness jr and outputs a proof string $\pi \in \mathbb{F}^m$. We shall assume this algorithm generates random coins by sampling uniformly from $\mathbb{F}^{pl}$.

- $\mathsf{FLP.Query}(x \in \mathbb{F}^n, \pi \in \mathbb{F}^m, \mathit{jr} \in \mathbb{F}^{jl}) \to \sigma \in \mathbb{F}^v$ is the randomized **query-generation** algorithm that takes in an input x, proof string π, and joint randomness jr and outputs a verifier string σ. We shall assume the random coins are sampled uniformly from $\mathbb{F}^{ql}$.

```
Algorithm $\mathsf{View}_{\mathsf{FLP}}(x)$:
  $jr \leftarrow\!\!\$\, \mathbb{F}^{jl}$; $qr \leftarrow\!\!\$\, \mathbb{F}^{ql}$
  $\pi \leftarrow\!\!\$\, \mathsf{Prove}(x, jr)$
  $\sigma \leftarrow \mathsf{Query}(x, \pi, jr;\ qr)$
  ret $jr \,\|\, qr \,\|\, \sigma$
```

```
Algorithm $\mathsf{Err}_{\mathsf{FLP}}(P^*)$:
  $(st_{P^*}, x) \leftarrow\!\!\$\, P^*(\,)$; $jr \leftarrow\!\!\$\, \mathbb{F}^{jl}$
  $\pi \leftarrow\!\!\$\, P^*(st_{P^*}, jr)$
  $\sigma \leftarrow\!\!\$\, \mathsf{Query}(x, \pi, jr)$
  ret $x \notin \mathcal{L} \wedge \mathsf{Decide}(\sigma)$
```

Figure 2: Procedures for defining security of FLPs.

- $\mathsf{FLP.Decide}(\sigma \in \mathbb{F}^v) \to acc \in \{0,1\}$ is the deterministic **decision predicate** that takes in a verifier string σ and outputs a bit acc indicating whether the input is valid.

We require the field $\mathbb{F}$ to have prime order; we occasionally denote its order by $\mathbb{F}.p$. We say that FLP is *fully linear* if the query-generation algorithm computes a linear function of the input and proof. That is, there exists a function Q whose output is a matrix in $\mathbb{F}^{v\times(n+m)}$ and, for all inputs x, proofs π, joint randomnesses jr, and query randomnesses qr, it holds that $\mathsf{Query}(x, \pi, jr;\ qr) = Q(jr;\ qr) \cdot (x \,\|\, \pi) \in \mathbb{F}^v$.

Associated with FLP is a language $\mathcal{L} \subseteq \mathbb{F}^n$. We say that FLP is **complete for $\mathcal{L}$** if the proof system outputs 1 whenever the input is in $\mathcal{L}$. That is, for all $x \in \mathcal{L}$ it holds that

$$\Pr\big[\,\mathsf{Decide}(\sigma) : jr \leftarrow\!\!\$\, \mathbb{F}^{jl}; \pi \leftarrow\!\!\$\, \mathsf{Prove}(x, jr); \sigma \leftarrow\!\!\$\, \mathsf{Query}(x, \pi, jr)\,\big] = 1\,.$$

We define soundness of FLP in terms of experiment $\mathsf{Err}_{\mathsf{FLP}}(P^*)$ shown in Figure 2 associated with a malicious prover P^*. In this experiment, the prover commits to an invalid input $x \in \mathbb{F}^n \setminus \mathcal{L}$. Next, joint randomness jr is generated and given to P^*, who then generates a proof π. Finally, the verifier is run on x, π, jr; the malicious prover "wins" if the verifier deems the input valid. We say FLP is **ϵ-sound for $\mathcal{L}$** if for all P^* it holds that $\Pr\big[\,\mathsf{Err}_{\mathsf{FLP}}(P^*)\,\big] \le \epsilon$.

Let $\mathsf{View}_{\mathsf{FLP}}(x)$ denote the procedure defined in Figure 2. We say FLP is **δ-statistical, strong, honest-verifier zero-knowledge**—or, simply, **δ-private**—if the verifier's view can be simulated without knowledge of the input. That is, there exists a randomized algorithm S such that for all $x \in \mathcal{L}$ it holds that

$$\sum_{\omega} \big|\Pr\big[\,\mathsf{View}_{\mathsf{FLP}}(x) = \omega\,\big] - \Pr\big[\,S(\,) = \omega\,\big]\big| \le \delta\,.$$

Comparison to Boneh et al. [17]. Our syntax diverges slightly from BBCG+19 in two main respects. First, we have tailored the syntax to 1.5-round, public-coin IOP systems (cf. [17, Section 3.2]), as this is the only type of system considered in the VDAF specification [9]. Following the spec, we refer to the "random challenge" as the "joint randomness", as this allows us to more easily distinguish the challenge from the randomness consumed locally by the prover and verifier. Second, following the VDAF specification [9], we have adapted the syntax so that it describes explicitly the computations of the prover and verifier. Namely, our query-generation algorithm takes in the input and proof and outputs the verifier string consumed by the decision algorithm, whereas in BBCG+19, the query-generation algorithm outputs a description of the linear function used to compute the verifier string.

Our notion of FLP soundness differs slightly from BBCG+19 in that it explicitly requires the prover to "commit" to the invalid prior to the joint randomness being generated. This clarifies that the joint randomness needs to be independent of the input in order for soundness to be achievable.

Incremental Distributed Point Functions. A point function is a function that is 0 everywhere except on a special input α; an incremental point function is a function that is 0 everywhere except on *any prefix of* α. One can imagine arranging the co-domain of this function into a complete, binary tree in which the nodes are labeled with prefixes; and for each node labeled p, its children are labeled with $p \,\|\, 0$ and $p \,\|\, 1$. Each node on the path to the leaf node α is assigned a non-0 value, and all other nodes are assigned 0. (See [18, Figure 4] for an illustration.)

An incremental point function that gives output $\vec{\beta}[\ell]$ on the length-ℓ prefix of α is defined formally as:

$$f_{\alpha,\vec{\beta}}\Big(pfx \in \{0,1\}^{\le\eta}\Big) = \begin{cases} \vec{\beta}[\ |pfx|\ ] & \text{if } pfx \text{ is a prefix of } \alpha \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

An *Incremental Distributed Point Function (IDPF)* [18] is a concise secret sharing of an incremental point function. We recall the definition of an IDPF from Boneh et al. [18] and restrict it slightly to suit the constructions of [9]. An IDPF's domain is the set of bitstrings of length at most η. For each input length ℓ, the IDPF generates outputs in the group $\mathbb{G}_\ell$. We present definitions only for the case of 2 parties, since leading constructions are specialized for that case. Let η, and κ be positive integers, let $\mathcal{M}$ be a set, and let $\mathbb{G}_\ell$ be a group for each $\ell \in [\eta]$. An IDPF is a pair of algorithms:

- $\mathsf{IDPF.Gen}(\alpha \in \{0,1\}^\eta, \vec{\beta} \in \mathbb{G}_1 \times \cdots \times \mathbb{G}_\eta) \to (\{0,1\}^\kappa)^2 \times \mathcal{M}$ is the **key generation** algorithm that takes a bitstring α and a vector $\vec{\beta}$ of point values, each of which is an element of the group $\mathbb{G}_\ell$ for the corresponding input length. It outputs a pair of key shares and a "public share" (an element of $\mathcal{M}$).
- $\mathsf{IDPF.Eval}(id \in \{1,2\}, key \in \{0,1\}^\kappa, pub \in \mathcal{M}, pfx \in \{0,1\}^\ell) \to \mathbb{G}_\ell$ is the **point-function evaluation** algorithm that takes in a shareholder index, an IDPF key share, a public share pub, and a prefix string of $\ell \leq \eta$ bits, then outputs a share of the IDPF output.

An IDPF is *correct* if for all $\alpha \in \{0,1\}^\eta$, all $\vec{\beta} \in \mathbb{G}_1 \times \cdots \times \mathbb{G}_\eta$, all $(key_1, key_2, pub) \in [\mathsf{IDPF.Gen}(\alpha, \vec{\beta})]$, and all strings pfx of length $\ell \leq \eta$:

$$f_{\alpha,\vec{\beta}}(pfx) = \sum_{\hat{\jmath} \in \{1,2\}} \mathsf{IDPF.Eval}(\hat{\jmath}, key_{\hat{\jmath}}, pub, pfx)\,.$$

We define *privacy* for an IDPF later in Section 5.1.

3 Security Model

3.1 Syntax

As discussed in Section 1, a VDAF can be thought of as a protocol for evaluating an aggregation function F that takes as input the vector of measurements generated by the clients and outputs an aggregate result. In addition, the function may include an auxiliary "aggregation parameter" that allows the measurements to be "refined" to contain only the information of interest to the collector. Accordingly, prior to executing the VDAF, each aggregator's state is initialized with this aggregation parameter.

Recall that execution of a VDAF proceeds in four distinct phases. (See Figure 1 for an illustration.) We formalize the computation of the parties in each phase as the component algorithms of a VDAF:

- $\mathsf{Shard}(m \in \mathcal{I}, n \in \mathcal{N}) \to (msg_{\text{Init}} \in \mathcal{M}, \vec{x} \in \mathcal{X}^s)$ is the randomized **sharding** algorithm run by the client. It takes in the client's input measurement m and a nonce n and returns an **initial message**[2] to be broadcasted to all aggregators and a sequence of **input shares**, one for each of the s aggregators.
- $\mathsf{Prep}(\hat{\jmath} \in [s], sk \in \mathcal{SK}, st \in \mathcal{Q}, n \in \mathcal{N}, \vec{msg} \in \mathcal{M}^*, x \in \mathcal{X}) \to (sts \in \{\texttt{running}, \texttt{finished}, \texttt{failed}\}, out \in (\mathcal{Q} \times \mathcal{M}) \cup \mathcal{Y} \cup \{\bot\}$ is the deterministic, interactive **preparation** algorithm run by each aggregator during the online preparation process. Its inputs are the share index $\hat{\jmath}$, the **verification key** shared by the aggregators sk, the current state st, the nonce n, the most recent round of **broadcast messages** $\vec{msg}$ (or $(msg_{\text{Init}},)$ if this is the first round), and the aggregator's input share x. The preparation algorithm returns an indication sts of whether the process is `running`, `finished`, or `failed`. When the status is `running`, the output includes the aggregator's next state and broadcast message $((st, msg) \in \mathcal{Q} \times \mathcal{M})$; and when the status is `finished`, the output includes the aggregator's **refined share** $(y \in \mathcal{Y})$.
- $\mathsf{Agg}(\vec{y} \in \mathcal{Y}^*) \to a \in \mathcal{A}$ is the deterministic *aggregation* algorithm run locally by each aggregator. It takes in a sequence of refined shares $\vec{y}$ and outputs an **aggregate share** a.
- $\mathsf{Unshard}(ct \in \mathbb{N}, \vec{a} \in \mathcal{A}^s) \to r \in \mathcal{O}$ is the deterministic *unsharding* algorithm used to compute the aggregate result r. Its inputs are the report count ct and aggregate shares $\vec{a}$.

[2] This message is called the "public share" in the specification.

The sets $\mathcal{I}$, $\mathcal{N}$, $\mathcal{M}$, $\mathcal{X}$, $\mathcal{SK}$, $\mathcal{Q}$, $\mathcal{Y}$, $\mathcal{A}$, and $\mathcal{O}$ must also be defined by the VDAF. (We typically do so only implicitly.) In addition to these sets, the VDAF specifies a set $\mathcal{Q}_{\text{Init}} \subseteq \mathcal{Q}$ of possible **initial states**.

Our security definitions for VDAFs require three additional syntactic properties. The first is a property we call **refinement consistency**. Intuitively, this property insists that, for a given initial state, the VDAF defines the set of refined measurements with respect to which the validity of the refined shares is to be verified. For Doplar for example (Section 5), the set of measurements are fixed-length bitstrings, while the refined measurements are one-hot vectors over a finite field. Formally, refinement consistency requires the existence of functions refine and refineFromShares such that for all m, n and $st_{\text{Init}} \in \mathcal{Q}_{\text{Init}}$,

$$\Pr[\mathsf{refine}(st_{\text{Init}}, m) = \mathsf{refineFromShares}(st_{\text{Init}}, msg, \vec{x}) : \\ (msg, \vec{x}) \leftarrow\!\!\$\ \mathsf{Shard}(m, n)] = 1\,.$$

Second, we require **aggregation consistency**, which means, roughly, that aggregating refined shares into aggregate shares, then unsharding, is equivalent to first unsharding the individual refined shares, then aggregating. To illustrate this idea, imagine arranging the refined shares into a matrix, where the rows correspond to aggregators and the columns to measurements. Aggregation consistency means that one can either add up the columns, then the rows, or add up the rows, then the columns. Formally, we require the existence of a function finishResult such that for all refined shares $y_1^1, \ldots, y_{ct}^1, \ldots, y_1^s, \ldots, y_{ct}^s \in \mathcal{Y}$, it holds that

$$\begin{aligned} &\mathsf{Unshard}(ct, (\mathsf{Agg}(y_1^1, \ldots, y_{ct}^1), \ldots, \mathsf{Agg}(y_1^s, \ldots, y_{ct}^s))) = \\ &\quad \mathsf{finishResult}(ct, \mathsf{Unshard}(1, (\mathsf{Agg}(y_1^1), \ldots, \mathsf{Agg}(y_1^s))), \\ &\qquad\qquad \ldots, \mathsf{Unshard}(1, (\mathsf{Agg}(y_{ct}^1), \ldots, \mathsf{Agg}(y_{ct}^s))))\,. \end{aligned}$$

We will see that these notions of refinement and aggregation consistency, while fairly technical in nature, are trivial to show for natural constructions (including Prio3 and Doplar).

Lastly, our privacy definition allows the VDAF to be executed multiple times over the same batch of measurements, each time beginning with a new initial state. (This accounts for the iterative nature of IDPFs.) Depending on the VDAF, it may be necessary for aggregators to restrict the sequence of initial states to prevent trivial leakage. Accordingly, we require each VDAF to specify an **allowed-state** algorithm validSt that takes in the sequence of previous initial states and the next initial state and returns a bit indicating whether the next initial state is allowed.

Remark 1. *A notable feature of the VDAF syntax is the "verification key" shared by the aggregators. Looking ahead, this key is used to derive, from the nonce supplied by the client, shared randomness used for verifying refined shares. This is how the authors of the VDAF spec [9] chose to instantiate the "ideal coin-flipping functionality" used in the descriptions of protocols in the papers on which the spec is based [23, 17, 18]. As we will see in the next section, the details to how this functionality is instantiated are crucial to the privacy and robustness of VDAFs.*

3.2 Security

Three definitions are given for VDAFs. The first, completeness, is used to specify correct evaluation of an aggregation function. The others, robustness and privacy, roughly correspond[3] to the notions of the same names from [23, Appendix A].

Security considerations for DAP [30]. Recall from the introduction that the DAP standard being developed by the PPM working group is designed to securely execute a VDAF in a real world network. Aspects of our security model can be thought of as abstracting away the functionality provided by DAP. As such, many of our modeling decisions here amount to requirements that the DAP protocol must fulfill. We will highlight some of these considerations throughout this section.

[3] We have not attempted to work out formal relationships between our definitions and those of Corrigan-Gibbs and Boneh [23]; whether our definitions, when restricted to the same class of protocols, are stronger, weaker, or equivalent is an open question.

Completeness. We require that, when executed honestly, the VDAF evaluates its aggregation function F correctly. We formalize non-adversarial execution of Π via procedure Run_{Π} in Figure 3. Along with the VDAF Π, this procedure is parameterized by an initial state st_{Init} with which to configure the aggregators and a sequence of measurements and nonces to process into an aggregate result.

Algorithm Run processes the measurements as illustrated in Figure 1. First, each measurement is sharded into input shares by the submitting client (line 4), then refined into a set of refined shares by the aggregators (5–16). Next, the refined shares recovered by each aggregator are combined into an aggregate share (18). Finally, the aggregate shares are combined by the collector into the aggregate result (19).

Definition 1 (Completeness)**.** *Let $F : \mathcal{Q}_{Init} \times \mathcal{I}^* \to \mathcal{O}$ be a function. We say that VDAF Π is* complete *for F if for all $\vec{m} \in \mathcal{I}^*$ and $\vec{n} \in \mathcal{N}^*$ for which $|\vec{m}| = |\vec{n}|$ and $st_{Init} \in \mathcal{Q}_{Init}$ it holds that*

$$\Pr\left[\, \mathsf{Run}_{\Pi}(st_{Init}, \vec{m}, \vec{n}) = F(st_{Init}, \vec{m}) \,\right] = 1\,,$$

where the probability is over the randomness of Run *and its subroutines. We say that Π is* **complete** *if it is complete for some function F.*

Robustness. We say that VDAF Π is robust if, when all of the aggregators execute the protocol correctly, "valid" refined measurements are correctly aggregated, while any "invalid" measurements are filtered out by the aggregators (with high probability). This property is captured via the game $\mathsf{Exp}^{\mathrm{robust}}_{\Pi}(A)$ defined in Figure 3. In this game the adversary, acting as a coalition of malicious clients, submits reports to the aggregators, eavesdrops on their communication, and observes the result of their computation. This functionality is modeled by the $\underline{\mathsf{Prep}}$ oracle, which the adversary may query any number of times. It controls the nonce and initial state for each trial, but its oracle queries are subject to the restriction that, for each distinct nonce, the sequence of initial states must be valid (according to the allowed-state algorithm $\mathsf{validSt}$).

Validity is defined in terms of the refinement-consistency algorithms (see Section 3.1). Let $\mathcal{V}_{st_{\mathrm{Init}}} = \{\mathsf{refine}_{st_{\mathrm{Init}}}(m) : m \in \mathcal{I}\}$ be the set of refined measurements for initial state st_{Init}. The adversary wins the robustness game if, when run on initial state st_{Init}, initial message msg_{Init}, and input shares $\vec{x}$, either: (1) an aggregator accepts a share of an invalid refined measurement, i.e., one of the aggregators ends in state `finished`, but the refined share y is not valid (i.e., not in the set $\mathcal{V}_{st_{\mathrm{Init}}}$, see line 15 in Figure 3); or (2) the refined shares computed by the aggregators do not match the expected refined measurement, i.e., unsharding the refined shares does not result in y (line 18).

Definition 2 (Robustness)**.** *Define the advantage of A in defeating the robustness of VDAF Π as*

$$\mathsf{Adv}^{\mathrm{robust}}_{\Pi}(A) = \Pr\left[\, \mathsf{Exp}^{\mathrm{robust}}_{\Pi}(A) \,\right].$$

Informally, we say that Π is **robust** *if for every efficient adversary A, the value of $\mathsf{Adv}^{\mathrm{robust}}_{\Pi}(A)$ is small.*

Remark 2. *If a VDAF is robust in the sense of Definition 2 and aggregation-consistent, then the VDAF is also robust in the sense of [23, Definition 6]. Namely, as long as the aggregators execute the VDAF correctly, the collector is guaranteed to correctly aggregate measurements from honest clients (and reject the measurements from dishonest clients). The aggregation function that is computed is determined by the* $\mathsf{finishResult}$ *function implied by aggregation consistency, namely $F(st_{Init}, m_1, \ldots, m_{ct}) = \mathsf{finishResult}(ct, (y_1, \ldots, y_{ct}))$, where $y_{\hat{k}}$ is the refined measurement obtained from refining $m_{\hat{k}}$ with st_{Init}.*

Privacy. We formalize privacy via the indistinguishability game $\mathsf{Exp}^{\mathrm{priv}}_{\Pi,t}(A)$ in the right panel of Figure 4. The game is associated with VDAF Π, adversary A, and **corruption threshold** t. We consider an attacker that controls the collector and statically corrupts at most t aggregators (lines 1–2). Using its $\underline{\mathsf{Prep}}$ oracle (lines 16–28), the adversary controls transmission of all messages in the protocol, *except* for the honestly generated input shares sent to honest (uncorrupted) aggregators. We assume that the adversary also controls setup (see the $\underline{\mathsf{Setup}}$ oracle on lines 11–15), meaning that it can pick the verification keys for honest aggregators (1) and the initial state of each run of the preparation phase (14). This captures the real-world setting of the DAP protocol [30], where one of the aggregators (the "leader") effectively picks these values on behalf of the others (the "helpers"). Note that our game requires the secret key to be committed to prior to generating

```
Algorithm $\mathsf{Run}_{\Pi}(st_{\text{Init}}, \vec{m}, \vec{n})$:
$sk \leftarrow\!\!\$\ \mathcal{SK}$; $ct \leftarrow |\vec{m}|$
// Shard/Prepare
for $\hat{k} \in [ct]$:
  $(msg, \vec{x}) \leftarrow \Pi.\mathsf{Shard}(\vec{m}[\hat{k}], \vec{n}[\hat{k}])$
  $\text{Msg}[0,1] \leftarrow msg$
  for $\hat{j} \in [s]$: $\text{St}[\hat{j}] \leftarrow st_{\text{Init}}$
  for $\hat{\ell} \in [r+1]$:
    for $\hat{j} \in [s]$:
      $(sts, out) \leftarrow \Pi.\mathsf{Prep}(\hat{j}, sk, \text{St}[\hat{j}],$
                        $\vec{n}[\hat{k}], \text{Msg}[\hat{\ell}\text{-}1, \cdot], \vec{x}[\hat{j}])$
      if $sts = \texttt{running}$:
        $(\text{St}[\hat{j}], msg) \leftarrow out$
        $\text{Msg}[\hat{\ell}, \hat{j}] \leftarrow msg$
      else if $sts = \texttt{finished}$:
        $\text{Out}[\hat{j}, \hat{k}] \leftarrow out$
      else if $sts = \texttt{failed}$: ret $\bot$
// Aggregate/Unshard
for $\hat{j} \in [s]$: $\vec{a}[\hat{j}] \leftarrow \Pi.\mathsf{Agg}(\text{Out}[\hat{j}, \cdot])$
ret $\Pi.\mathsf{Unshard}(ct, \vec{a})$
```

```
Game $\mathsf{Exp}_{\Pi}^{\text{robust}}(A)$:
$sk \leftarrow\!\!\$\ \mathcal{SK}$; $w \leftarrow \texttt{false}$; $A^{\underline{\mathsf{Prep}}}()$; ret $w$

$\underline{\mathsf{Prep}}(n \in \mathcal{N}, \vec{x} \in \mathcal{X}^s, msg_{\text{Init}} \in \mathcal{M}, st_{\text{Init}} \in \mathcal{Q}_{\text{Init}})$:
if not $\Pi.\mathsf{validSt}(\text{Used}[n], st_{\text{Init}})$: ret $\bot$
$\text{Used}[n] \leftarrow \text{Used}[n] \,\|\, (st_{\text{Init}}, )$
$\text{Msg}[0,1] \leftarrow msg_{\text{Init}}$
$y \leftarrow \Pi.\mathsf{refineFromShares}(st_{\text{Init}}, msg_{\text{Init}}, \vec{x})$
for $\hat{j} \in [s]$: $\text{St}[\hat{j}] \leftarrow st_{\text{Init}}$
for $\hat{\ell} \in [r+1]$:
  for $\hat{j} \in [s]$:
    $(sts, out) \leftarrow \Pi.\mathsf{Prep}(\hat{j}, sk, \text{St}[\hat{j}]$
                    $n, \text{Msg}[\hat{\ell}\text{-}1, \cdot], \vec{x}[\hat{j}])$
    if $sts = \texttt{running}$:
      $(\text{St}[\hat{j}], msg) \leftarrow out$
      $\text{Msg}[\hat{\ell}, \hat{j}] \leftarrow msg$
    else if $sts = \texttt{finished}$:
      $y_{\hat{j}} \leftarrow out$; $\tilde{w} \leftarrow [y \notin \mathcal{V}_{st_{\text{Init}}}]$
    else if $sts = \texttt{failed}$: pass
if not $\tilde{w}$:
  $\tilde{w} \leftarrow [y \neq \Pi.\mathsf{Unshard}(1, (\Pi.\mathsf{Agg}(y_{\hat{j}}))_{\hat{j} \in s}]$
$w \leftarrow w \vee \tilde{w}$; ret $(w, \text{Msg})$
```

Figure 3: Left: Procedure for defining completeness of r-round, s-party VDAF Π. Right: Game for defining robustness of Π. Let $\mathcal{Q}_{\text{Init}} \subseteq \mathcal{Q}$ denote the set of valid initial states and, for each $st_{\text{Init}} \in \mathcal{Q}_{\text{Init}}$, let $\mathcal{V}_{st_{\text{Init}}} = \{\mathsf{refine}_{st_{\text{Init}}}(m) : m \in \mathcal{I}\}$.

measurements: this is a deliberate restriction that was necessary to prove security of our constructions. (It is necessary for DAP to enforce this restriction.)

The initial state for each run is subject to the restriction imposed by the allowed-state algorithm defined by the VDAF (lines 11–13). (Accordingly, it is necessary for honest aggregators to enforce this restriction in the DAP protocol.)

The game asks A to distinguish execution of the protocol on two sets of measurements of its choosing. To capture this, the attacker is given an oracle $\underline{\mathsf{Shard}}$ (lines 6–10) that models execution of the honest clients. This oracle takes in two measurements m_0, m_1 and shards m_b, where b is the challenge bit chosen at the start of the game, and returns the initial message and the input shares of the corrupted aggregators. The oracle chooses a nonce n from the nonce space $\mathcal{N}$ at random. (Accordingly, the DAP protocol must arrange for clients to choose their nonces at random.)

To model an attacker that controls the collector, the game allows the adversary to learn the aggregate shares computed by honest aggregators. This is captured by the $\underline{\mathsf{Agg}}$ oracle (lines 29–35). Queries to this oracle are subject to the restriction that the aggregate share does not trivially leak the challenge bit: namely, the aggregate of both batches of measurements specified by the adversary must be equal (31). (Tables $\text{Batch}_0, \text{Batch}_1$ keep track of the pairs of measurements m_0, m_1 passed to the $\underline{\mathsf{Shard}}$ for which a given aggregator has recovered a refined share for a given initial state.) This restriction is analogous to the "leakage function" provided to the simulator in previous simulation-style definitions. See [23, Appendix A] and [18, Appendix A]. We consider something slightly stronger: if the honest aggregators disagree either on the initial state or the verification key, then we do not impose the restriction (32). This amounts to demanding that the aggregate shares leak nothing in this case.

Definition 3 (Privacy)**.** *Let Π be an s-party VDAF and let $t < s$ be a positive integer. Define the t-advantage of A in attacking the privacy of Π as*

$$\mathsf{Adv}_{\Pi,t}^{\text{priv}}(A) = 2 \cdot \Pr\left[\, \mathsf{Exp}_{\Pi,t}^{\text{priv}}(A) \,\right] - 1\,.$$

*Informally, we say that Π is t-**private** if for every efficient A the value of $\mathsf{Adv}_{\Pi,t}^{\text{priv}}(A)$ is small.*

```
Game $\mathsf{Exp}^{\mathrm{priv}}_{\Pi,t}(A)$:
  $(st_A, \mathcal{V}, (sk_{\hat{j}})_{\hat{j}\in\mathcal{V}}) \leftarrow\$ A(\,)$
  if $|\mathcal{V}| + t \neq s$ return $\bot$
  $b \leftarrow\$ \{0,1\}$
  $b' \leftarrow\$ A^{\underline{\mathsf{Shard}},\underline{\mathsf{Setup}},\underline{\mathsf{Prep}},\underline{\mathsf{Agg}}}(st_A)$
  ret $b = b'$

Shard($\hat{k} \in \mathbb{N}, m_0, m_1 \in \mathcal{I}$):
  if $\mathrm{Used}[\hat{k}] \neq \bot$: ret $\bot$
  $n \leftarrow\$ \mathcal{N}$
  $(\mathrm{Pub}[\hat{k}], \mathrm{In}[\hat{k}, \cdot]) \leftarrow\$ \Pi.\mathsf{Shard}(m_b, n)$
  $\mathrm{Used}[\hat{k}] \leftarrow (n, m_0, m_1)$
  ret $(n, \mathrm{Pub}[\hat{k}], (\mathrm{In}[\hat{k}, \hat{j}])_{\hat{j}\in\mathcal{T}})$

Setup($\hat{\imath} \in \mathbb{N}, \hat{j} \in \mathcal{V}, st_{\mathrm{Init}} \in \mathcal{Q}_{\mathrm{Init}}$):
  if $\mathrm{Status}[\hat{\imath}, \hat{j}] \neq \bot$
      or not $\Pi.\mathsf{validSt}(\mathrm{Setup}[\cdot, \hat{j}], st_{\mathrm{Init}})$:
    ret $\bot$
  $\mathrm{Setup}[\hat{\imath}, \hat{j}] \leftarrow st_{\mathrm{Init}}$
  $\mathrm{Status}[\hat{\imath}, \hat{j}] \leftarrow$ running

Prep($\hat{\imath} \in \mathbb{N}, \hat{j} \in \mathcal{V}, \hat{k} \in \mathbb{N}, \vec{msg} \in \mathcal{M}^*$):
  if $\mathrm{Status}[\hat{\imath}, \hat{j}] \neq$ running or $\mathrm{In}[\hat{k}, \hat{j}] = \bot$: ret $\bot$
  if $\mathrm{St}[\hat{\imath}, \hat{j}, \hat{k}] = \bot$:
    $\mathrm{St}[\hat{\imath}, \hat{j}, \hat{k}] \leftarrow \mathrm{Setup}[\hat{\imath}, \hat{j}]$; $\vec{msg} \leftarrow (\mathrm{Pub}[\hat{k}],)$
  $(n, m_0, m_1) \leftarrow \mathrm{Used}[\hat{k}]$
  $(sts, out) \leftarrow$
      $\Pi.\mathsf{Prep}(\hat{j}, sk_{\hat{j}}, \mathrm{St}[\hat{\imath}, \hat{j}, \hat{k}], n, \vec{msg}, \mathrm{In}[\hat{k}, \hat{j}])$
  if $sts =$ running:
    $(st, msg) \leftarrow out$; $\mathrm{St}[\hat{\imath}, \hat{j}, \hat{k}] \leftarrow st$
  else if $sts =$ finished:
    $\mathrm{St}[\hat{\imath}, \hat{j}, \hat{k}] \leftarrow \bot$; $\mathrm{Out}[\hat{\imath}, \hat{j}, \hat{k}] \leftarrow out$
    $\mathrm{Batch}_0[\hat{\imath}, \hat{j}, \hat{k}] \leftarrow m_0$; $\mathrm{Batch}_1[\hat{\imath}, \hat{j}, \hat{k}] \leftarrow m_1$
  else if $sts =$ failed: $\mathrm{St}[\hat{\imath}, \hat{j}, \hat{k}] \leftarrow \bot$
  ret $(sts, msg)$

Agg($\hat{\imath} \in \mathbb{N}, \hat{j} \in \mathcal{V}$):
  if $\mathrm{Status}[\hat{\imath}, \hat{j}] \neq$ running: ret $\bot$
  $(st_1, \ldots, st_s) \leftarrow \mathrm{Setup}[\hat{\imath}, \cdot]$
  if $F(st_{\hat{j}}, \mathrm{Batch}_0[\hat{\imath}, \hat{j}, \cdot]) \neq F(st_{\hat{j}}, \mathrm{Batch}_1[\hat{\imath}, \hat{j}, \cdot])$
      and $(\forall j, j' \in \mathcal{V})\ st_j = st_{j'} \wedge sk_j = sk_{j'}$:
    ret $\bot$
  $\mathrm{Status}[\hat{\imath}, \hat{j}] \leftarrow$ finished
  ret $\Pi.\mathsf{Agg}(\mathrm{Out}[\hat{\imath}, \hat{j}, \cdot])$
```

Figure 4: Game for defining privacy of a complete, s-party VDAF Π for corruption threshold ≥ 0. Let F denote the aggregation function for which Π is complete and let $\mathcal{Q}_{\mathrm{Init}}$ its set of initial states. Let $\mathcal{T} = [s] \setminus \mathcal{V}$.

4 Prio3

In this section we present our security analysis for Prio3, one of the candidates for standardization specified in draft-irtf-cfrg-vdaf-03 [9]. The starting point for this VDAF is an FLP system (Section 2) that defines the set of valid measurements. Drawing on techniques from Boneh et al. [17], Prio3 exploits the full-linearity property to allow the aggregators to validate the secret shared input. However, in order for the resulting VDAF to be suitable for a particular aggregation function $F : \mathcal{I} \to \mathcal{O}$, we need the proof system to define how measurements ($\mathcal{I}$) are encoded as inputs to the prover and how refined shares are processed into the aggregate results ($\mathcal{O}$).

Definition 4 (Affine, aggregatable encodings [23, Sec. 5.])**.** *Let $F : \mathcal{I} \to \mathcal{O}$ be a function. An FLP system* FLP *admits an* affine, aggregatable encoding *for F if it defines the following algorithms:*

- $\mathsf{FLP}.\mathsf{Encode}(m \in \mathcal{I}) \to inp \in \mathbb{F}^n$ *is an injective map from the domain of F to the input space $\mathbb{F}^n$ of* FLP*.*
- $\mathsf{FLP}.\mathsf{Truncate}(inp \in \mathbb{F}^n) \to out \in \mathbb{F}^{ol}$ *refines an FLP input into a format suitable for aggregation. We call ol the* output length*.*
- $\mathsf{FLP}.\mathsf{Decode}(ct \in \mathbb{N}, out \in \mathbb{F}^{ol}) \to a \in \mathcal{O}$ *converts a refined, aggregated output out to its final form a. This computation may depend on the number of measurements ct.*

Correctness requires that for all $ct \geq 0$ and $\vec{m} \in \mathcal{I}^{ct}$ it holds that

$$F(\vec{m}\,) = \mathsf{Decode}\Big(ct, \sum_{i\in[ct]} \mathsf{Truncate}\left(\mathsf{Encode}\left(\vec{m}[i]\right)\right)\Big)\,.$$

Let FLP be an FLP system that admits an affine, aggregatable encoding for F and let PRG be a PRG. We specify the core algorithms of $\mathsf{Prio3}[\mathsf{FLP}, \mathsf{PRG}]$ in Figure 5. (This version includes changes to draft-irtf-cfrg-vdaf-03 [9], as we discuss below.) The sharding algorithm begins by encoding the measurement as

```
Algorithm $\mathsf{Shard}(m, n)$:
$\mathit{inp} \leftarrow \mathsf{Encode}(m)$
for $\hat{j} \in [2..s]$:
    $\mathit{blind}_{\hat{j}}, \mathit{xseed}_{\hat{j}}, \mathit{pseed}_{\hat{j}} \leftarrow\!\!\$\ \{0,1\}^{\kappa}$
    $\vec{x}[\hat{j}] \leftarrow \mathsf{RG}_2(\mathit{xseed}_{\hat{j}}, \hat{j})$
    $\vec{\mathit{rseed}}[\hat{j}] \leftarrow \mathsf{RG}_7(\mathit{blind}_{\hat{j}}, \hat{j} \parallel n \parallel \vec{x}[\hat{j}])$
$\vec{x}[1] \leftarrow \mathit{inp} - \sum_{\hat{j}=2}^{s} \vec{x}[\hat{j}]$
$\mathit{blind}_1 \leftarrow\!\!\$\ \{0,1\}^{\kappa}$
$\vec{\mathit{rseed}}[1] \leftarrow \mathsf{RG}_7(\mathit{blind}_1, 1 \parallel n \parallel \vec{x}[1])$
$\mathit{jseed} \leftarrow \mathsf{RG}_6(0^{\kappa}, \vec{\mathit{rseed}})$; $\mathit{jr} \leftarrow \mathsf{RG}_1(\mathit{jseed}, \varepsilon)$
$\mathit{ps} \leftarrow\!\!\$\ \{0,1\}^{\kappa}$; $\mathit{pr} \leftarrow \mathsf{RG}_4(\mathit{ps}, \varepsilon)$
$\vec{\pi}[1] \leftarrow \mathsf{Prove}(\mathit{inp}, \mathit{jr}\,;\, \mathit{pr})$
$\vec{\pi}[1] \leftarrow \vec{\pi}[1] - \sum_{\hat{j}=2}^{s} \mathsf{RG}_3(\mathit{pseed}_{\hat{j}}, \hat{j})$
$\vec{x}[1] \leftarrow (\vec{x}[1], \vec{\pi}[1], \mathit{blind}_1)$
for $\hat{j} \in [2..s]$:
    $\vec{x}[\hat{j}] \leftarrow (\mathit{xseed}_{\hat{j}}, \mathit{pseed}_{\hat{j}}, \mathit{blind}_{\hat{j}})$
ret $(\vec{\mathit{rseed}}, \vec{x})$
```

```
Algorithm $\mathsf{Unpack}(\hat{j}, x)$:
if $\hat{j} = 1$: $(\mathit{inp}, \pi, \mathit{blind}) \leftarrow x$
else:
    $(\mathit{xseed}, \mathit{pseed}, \mathit{blind}) \leftarrow x$
    $\mathit{inp} \leftarrow \mathsf{RG}_2(\mathit{xseed}, \hat{j})$
    $\pi \leftarrow \mathsf{RG}_3(\mathit{pseed}, \hat{j})$
ret $(\mathit{inp}, \pi, \mathit{blind})$
```

```
Algorithm $\mathsf{Prep}(\hat{j}, \mathit{sk}, \mathit{st}, n, \vec{\mathit{msg}}, x)$:
if $\mathit{st} = \varepsilon$:   // Process initial message from client
    $(\mathit{inp}, \pi, \mathit{blind}) \leftarrow \mathsf{Unpack}(\hat{j}, x)$
    $(\vec{\mathit{rseed}}, ) \leftarrow \vec{\mathit{msg}}$; $\vec{\mathit{rseed}}[\hat{j}] \leftarrow \mathsf{RG}_7(\mathit{blind}, \hat{j} \parallel n \parallel \mathit{inp})$
    $\mathit{jseed} \leftarrow \mathsf{RG}_6(0^{\kappa}, \vec{\mathit{rseed}})$; $\mathit{jr} \leftarrow \mathsf{RG}_1(\mathit{jseed}, \varepsilon)$
    $\mathit{qr} \leftarrow \mathsf{RG}_5(\mathit{sk}, n)$
    $\mathit{msg} \leftarrow (\mathsf{Query}(\mathit{inp}, \pi, \mathit{jr}; \mathit{qr}), \vec{\mathit{rseed}}[\hat{j}])$
    $\mathit{st} \leftarrow (\mathit{jseed}, \mathsf{Truncate}(\mathit{inp}))$
    ret $(\texttt{running}, \mathit{st}, \mathit{msg})$
// Process broadcast messages from aggregators
$(\mathit{jseed}, y) \leftarrow \mathit{st}$; $(\vec{\mathit{vfs}}[\hat{j}], \vec{\mathit{rseed}}[\hat{j}])_{\hat{j} \in [s]} \leftarrow \vec{\mathit{msg}}$
$\mathit{acc} \leftarrow \mathsf{Decide}(\sum_{\hat{j}=1}^{s} \vec{\mathit{vfs}}[\hat{j}])$
if $\mathit{acc}$ and $\mathit{jseed} = \mathsf{RG}_6(0^{\kappa}, \vec{\mathit{rseed}})$: ret $(\texttt{finished}, y)$
else ret $(\texttt{failed}, \bot)$
```

```
Algorithm $\mathsf{Agg}(\vec{y})$:
ret $\sum_{i=1}^{|\vec{y}|} \vec{y}[i]$
```

```
Algorithm $\mathsf{Unshard}(\mathit{ct}, \vec{a})$:
ret $\mathsf{Decode}(\mathit{ct}, \sum_{i=1}^{|\vec{a}|} \vec{a}[i])$
```

```
Algorithm $\mathsf{RG}_i(\mathit{seed}, \mathit{cntxt})$:
$l \leftarrow (\mathit{jl}, n, m, \mathit{pl}, \mathit{ql})$
if $i \leq 5$: ret $\mathsf{Expand}[\mathsf{PRG}](\mathit{seed}, \texttt{label}_i \parallel \mathit{cntxt}, \mathbb{F}.p, l[i])$
else: ret $\mathsf{PRG.Next}(\mathsf{PRG.Init}(\mathit{seed}, \texttt{label}_i \parallel \mathit{cntxt}), \kappa)$
```

Figure 5: Definition of 1-round, s-party VDAF Prio3[FLP, PRG]. Let $\texttt{label}_1, \ldots, \texttt{label}_7$ be arbitrary, distinct bitstrings.

prescribed by the FLP. It then splits the encoded measurement *inp* into shares, generates a proof of *inp*'s validity, and splits the proof into shares as well. The joint randomness *jr* passed to the proof generation algorithm is derived from the input shares following the Fiat-Shamir-style transform described—but not formally analyzed—in [17, Section 6.2.3]. During preparation, the aggregators collectively re-compute *jr* from their input shares. Each aggregator broadcasts a share of the verifier by running the FLP query-generation algorithm on its share of the input and proof. (The query randomness *qr* is derived from the shared verification key *sk* and the nonce *n* provided by the environment.) The FLP decision algorithm is run on the combined verifier shares.

The aggregators must derive the joint randomness prior to computing their verifier shares. In order to allow them to perform both computations in parallel in a single round, the client sends in its initial message the sequence $\vec{\mathit{rseed}}$ of "joint randomness parts" consisting of the intermediate values computed by the aggregators. This allows *jr* to be computed immediately on receipt of the input shares. To detect if a malicious client transmitted malformed parts, the aggregators also verify the joint randomness was computed properly in the same flow.

Allowed initial states. The set of initial states for Prio3 is simply $\mathcal{Q}_{\text{Init}} = \{\varepsilon\}$. In our security analysis, we assume honest aggregators process a batch at most once. Accordingly, the allowed-state algorithm Prio3[FLP, PRG].validSt accepts only if the batch was not aggregated previously.

Consistency. The set of refined measurements includes any output of the affine, aggregatable encoding for FLP. On input of $st_{\text{Init}} \in \{\varepsilon\}$ and $m \in \mathcal{I}$, the refinement algorithm Prio3[FLP, PRG].refine first encodes m, then truncates and decodes it as prescribed by FLP. The refine-from-shares algorithm, refineFromShares, unpacks each input share (see Unpack in Figure 5), extracts the shares of the FLP input, truncates them, adds them together, and decodes the result.

For aggregation consistency, we require the encoding scheme for FLP to be aggregation-consistent in a similar sense. Specifically, there must exist a function $\mathsf{finishResult}$ such that for all outputs $out_1, \ldots, out_{ct} \in \mathbb{F}^{ol}$ it holds that $\mathsf{Decode}(ct, \sum_{\hat{k}\in[ct]} out_{\hat{k}}) = \mathsf{finishResult}(ct, \mathsf{Decode}(1, out_1), \ldots, \mathsf{Decode}(1,\ out_{ct}))$.

Changes to the specification [9]. Figure 5 differs from draft-03 of the VDAF spec in three ways. The most important change is to incorporate the nonce provided by the environment into the joint randomness computation. This turns out to be crucial for a tight robustness bound; without this change, we must contend with cases in which joint randomness is reused across reports.

Second, we have revised the domain separation tags for the PRG invocations so that each RG_i in Figure 5 can be treated as an independent random oracle.

Lastly, we have moved the joint randomness parts from the input shares into the client's initial broadcast message. This change allowed us to simplify our proofs somewhat, but we do not believe it is essential for security. It also has the added benefit of reducing overall communication overhead for $s > 2$.

Security. Fix $s > 2$ and let $\Pi = \mathsf{Prio3}[\mathsf{FLP}, \mathsf{PRG}]$ be as specified above. Let $\mathcal{N}$ denote the nonce space for Π and let κ denote the seed length of PRG.

Theorem 1. *Modeling each RG_i in Figure 5 as a random oracle, if FLP is ϵ-sound (Section 2), then for every adversary A against the robustness of Π it holds that*

$$\mathsf{Adv}_{\Pi}^{\text{robust}}(A) \leq (q_{\mathsf{RG}} + q_{\mathsf{Prep}}) \cdot \epsilon + \frac{q_{\mathsf{RG}} + q_{\mathsf{Prep}}{}^2}{2^{\kappa-1}} \ ,$$

where A makes q_{Prep} queries to $\underline{\mathsf{Prep}}$ and a total of q_{RG} queries to its random oracles.

For reasonable choices of the PRG seed size, the loosest term in this bound is $(q_{\mathsf{RG}} + q_{\mathsf{Prep}}) \cdot \epsilon$. The multiplicative loss of $q_{\mathsf{RG}} + q_{\mathsf{Prep}}$ reflects the adversary's ability to partially control the randomness of the FLP insofar as it is able to use rejection sampling to obtain query and joint randomness with any property. The ϵ-soundness of FLP bounds the probability of violating soundness in a single interaction, but in a VDAF the attacker may interact with the underlying FLP once in each of its q_{Prep} queries to $\underline{\mathsf{Prep}}$, and it can use its queries to RG_1 to bias these interactions' joint randomness.

Proof sketch. We sketch the security reduction here and defer the detailed proof to Appendix C.1. Our goal is to construct from A a malicious prover P^* for the soundness of FLP. The overall idea is to run A in a simulation of the robustness game for Π in which P^*'s instance of the soundness experiment (Figure 2) is embedded in a random $\underline{\mathsf{Prep}}$ query so that P^* wins its game precisely when A sets $w \leftarrow \texttt{true}$ for the first time in that query. The main difficulty is that P^* must arrange to use the joint randomness it received as input in its own game. To provide a consistent simulation of RG_1, we need to arrange to extract the input to commit to from A's queries. This results in a union bound over all queries to RG_1, either by the simulation of $\underline{\mathsf{Prep}}$ or by A directly. □

Remark 3. *For FLPs that do not make use of joint randomness (i.e., those for which $jl = 0$), queries to RG_1 can be disregarded, as this oracle is not used by Π. In particular, a similar reduction can be shown that results in a multiplicative loss of just q_{Prep}.*

Remark 4. *Although we have not addressed this explicitly in our specification, the extraction step of our security reduction relies on the encoding of the context string passed to each RG_i being invertible. (Similarly for Theorem 3.)*

Theorem 2. *Modeling each RG_i in Figure 5 as a random oracle, if FLP is δ-private, then for all $0 < t < s$ and attackers A it holds that*

$$\mathsf{Adv}_{\Pi,t}^{\text{priv}}(A) \leq 2q_{\mathsf{Shard}} \left(\delta + \frac{q_{\mathsf{RG}} + q_{\mathsf{Shard}}}{|\mathcal{N}|} + \frac{s \cdot q_{\mathsf{RG}}}{2^{\kappa-1}} \right) \ ,$$

where A makes q_{Shard} queries to $\underline{\mathsf{Shard}}$ and a total of q_{RG} queries to the random oracles.

Proof sketch. The full proof is given in Section C.2. The main idea is to arrange for A's queries to its oracles to be independent of the challenge bit. We do so via a game-playing argument in which we incrementally revise the game until the outcome of each oracle is independent of the current state of the game. The last step involves a hybrid argument, where in each hybrid world we replace one invocation of the proof- and query-generation algorithms of FLP (see Figure 2) with invocation of the simulator hypothesized by the δ-privacy of FLP. This accounts for the multiplicative loss of q_{Shard} in the bound. □

Remark 5. *Instead of using separate seeds for the input share, proof share, and blind, it may be safe to reuse the same seed for all three purposes, similar to the seed in Doplar (Section 5). This may result in a slightly looser bound: such a change would enable the attacker to test guesses of the input share because the known joint randomness part would be derived from the same seed.*

5 Doplar

In this section we describe and analyze Doplar, our round-reduced variant of Poplar1 [9]. Poplar1 is a candidate for standardization in draft-irtf-cfrg-vdaf-03; Doplar is introduced by our paper.

Poplar1 is designed to solve the "heavy hitters" problem (as described in Section 1) using an IDPF (Section 2) in the following way. Two aggregators hold shares of an IDPF key generated by the clients. Each evaluates its IDPF key at a number of equal-length candidate prefixes. They expect that the output is non-zero for at most one of these candidates; to verify this, they execute an MPC to determine if they hold shares of a one-hot vector, and that the non-zero value is in the desired range (i.e., equal to one or zero). If verification succeeds, then each adds its share of the vector together with the other verified shares. The result is a vector representing the number of measurements prefixed by each candidate.

The "secure sketch" MPC of Boneh et al. [18] requires two rounds of communication between the aggregators. (Computing and verifying this sketch occurs during the preparation phase of VDAF evaluation.) In this section we propose an alternative strategy that, leveraging techniques in Section 4, requires just one.

Our first step is to factor the validity check into two, parallelizable computations. The first computation is solely responsible for checking that the vector of IDPF outputs is one-hot. In Section 5.1 we extend IDPFs (Section 2) into *verifiable* IDPFs (VIDPFs), which preserve the same privacy properties as IDPFs, but additionally verify the one-hotness of the refined shares. In Appendix A we show how to instantiate this primitive using a simple technique from de Castro and Polychroniadou [22].

The second computation checks that the *sum* of the elements of the vector is in the desired range. Our first idea is to perform this range check using an FLP (Section 2). This does not work, however, since a standard FLP requires the prover to know the statement it is proving; in our case, it does not know the value of the sum computed by the aggregators, since it does not know the candidate prefixes. To overcome this, we show how to transform an FLP into one that is *delayed input* [43]. Such a proof system allows a proof to be generated for a *set* of potential inputs such that the honest verifier accepts the proof for any input in this set, but rejects otherwise (with high probability). We define delayed-input FLP in Section 5.2 and defer the construction to Appendix B.

The result is the 1-round, 2-party VDAF presented in Section 5.3. The cost of this round reduction is a modest increase in overall communication cost and CPU time, at least for the current instantiations of the VIDPF and delayed-input FLP. We compare the cost of Doplar and Poplar1 at the end of this section.

5.1 Verifiable IDPF

A **verifiable** IDPF (VIDPF) allows the dealer to prove to the shareholders that their shares represent a one-hot vector. For our purposes, we define a **one-hot vector** as a vector that is nonzero in *at most* one component (i.e., the all-zeroes vector is also one-hot). Verifiable function secret sharings (of which VIDPF is a special case) were previously considered in [19, 22], and a construction specifically for VIDPF was given in [22].

A VIDPF has two algorithms in addition to the usual $\mathsf{Gen}, \mathsf{Eval}$:

- $\mathsf{VIDPF.VEval}(id \in \{1,2\}, key \in \{0,1\}^\kappa, pub \in \mathcal{M},$ $\vec{x} \in (\{0,1\}^\ell)^u) \to \{0,1\}^* \times (\mathbb{G}_\ell)^u$ takes as input an IDPF share (private and public parts), and a sequence of IDPF inputs. It outputs a **verification value** and a sequence of output shares.

- $\mathsf{VIDPF.Verify}(h_1, h_2) \to \{0,1\}$ takes as input two verification values and returns a boolean.

We also overload the syntax of the plaintext evaluation function to take a vector of inputs, i.e., we let

$$f_{\alpha,\vec{\beta}}(\vec{x}) = \Big(f_{\alpha,\vec{\beta}}(\vec{x}[1]), f_{\alpha,\vec{\beta}}(\vec{x}[2]), \dots \Big).$$

We say VIDPF is *correct* if, for all $\alpha \in \{0,1\}^\eta$, all $\vec{\beta} \in \mathbb{G}_1 \times \cdots \times \mathbb{G}_\eta$, all $\vec{x} \in (\{0,1\}^\ell)^*$, all $(key_1, key_2, pub) \in [\mathsf{Gen}(\alpha, \vec{\beta})]$, all $(h_1, \vec{y}_1) \in [\mathsf{VEval}(1, key_1, pub, \vec{x})]$, and all $(h_2, \vec{y}_2) \in [\mathsf{VEval}(2, key_2, pub, \vec{x})]$:

- $\vec{y}_1 + \vec{y}_2 = f_{\alpha,\vec{\beta}}(\vec{x})$
- If $(\vec{y}_1 + \vec{y}_2)$ is a one-hot vector then $\mathcal{V}.\mathsf{Verify}(h_1, h_2) = 1$

Theorem 3 requires VIDPF to be *extractable*. Intuitively, there should be an algorithm that can extract $\alpha, \vec{\beta}$ from adversarially generated VIDPF key shares. Then VEval must produce shares consistent with the incremental point function $f_{\alpha,\vec{\beta}}$, whenever Verify succeeds. (A similar property is formalized for IDPFs by BBCG+21.) This property implies, among other things, that if Verify succeeds, then shareholders are guaranteed to hold shares of a one-hot vector. We formalize this property below.

Definition 5 (Extractable VIDPF (cf. [18, Definition 7]))**.** *Suppose that* VIDPF *is defined in terms of a random oracle with co-domain* $\mathcal{Y}$*. Refer to the game in Figure 6 associated to* VIDPF*,* **extractor** E*, and adversary* A*. Define* A*'s advantage in* **fooling** E *as* $\mathsf{Adv}^{\text{extract}}_{\mathsf{VIDPF},E}(A) = 2 \cdot \Pr\big[\, \mathsf{Exp}^{\text{extract}}_{\mathsf{VIDPF},E}(A) \,\big] - 1$.

Finally, our privacy reduction for Doplar (Theorem 4) requires the underlying VIDPF to be *private*, in the sense that one shareholder's view—consisting of its share $key_{\hat{j}}$, the public share pub, and the other shareholder's verification value h—leaks nothing about the secrets α and β. Prior definitions of verifiable FSS—e.g., the one in de Castro and Polychroniadou [22]—only define privacy with respect to a single vector of evaluation points and verification predicate, both of which are assumed to be known at the time of share generation. In our setting, shares are generated and only later is there a choice of evaluation points and verification predicates. The same shares may be evaluated many times, on different input vectors and with different verification predicates. This leads to a more interactive, and stronger, definition than in prior works.[4]

Definition 6. *Let* $\mathsf{Exp}^{\text{priv}}_{\mathsf{VIDPF},S}(A)$ *be the privacy game for* VIDPF*,* **simulator** $S = (S_1, S_2)$*, and adversary* A *defined in Figure 6. Define the advantage of* A *in distinguishing* S*'s simulation from its view of* VIDPF*'s execution as* $\mathsf{Adv}^{\text{priv}}_{\mathsf{VIDPF},S}(A) = 2 \cdot \Pr[\mathsf{Exp}^{\text{priv}}_{\mathsf{VIDPF},S}(A)] - 1$.

If this privacy game withholds the $\underline{\mathsf{Sketch}}$ oracle from the adversary (shaded in Figure 6) then we obtain the privacy game for plain IDPFs, with the adversary's advantage defined analogously.

In Appendix A we describe a VIDPF construction that satisfies all the necessary security properties. The construction is heavily based on the verifiable DPF technique from [22].

5.2 Delayed-Input FLPs

We introduce a new variant of fully linear proofs (FLPs), in which the prover does not know in advance which instance (i.e., input) will be used during verification. Instead, the proof is generated only knowing a set of possible instances; later, the proof is verified using one of those instances. For technical reasons, the proof and verification steps operate not on the instance, but on a *randomized encoding* of the instance. This extra randomness is useful in our eventual construction (Appendix B).

We adopt the terminology of **delayed-input**, which is standard in the study of (interactive) zero-knowledge protocols. In an interactive protocol with delayed input, the instance and witness need not be known/chosen until some intermediate round (often the prover's final round). In our setting, the actual choice of instance/witness is not chosen until after the prover finishes "speaking". The protocol of Lapidot and Shamir [43] is often regarded as the first ZK protocol with delayed input, while Katz and Ostrovsky [41] were the first to explicitly rely on the delayed input property while using a ZK proof in an application.

[4] The game does not need to provide an oracle for $\mathsf{VIDPF.Verify}$ since it is a deterministic algorithm whose inputs are known to the adversary.

```
Game $\mathsf{Exp}^{\mathrm{extract}}_{\mathsf{VIDPF},E}(A)$:
  $b \leftarrow\$ \{0,1\}$; $(key_1, key_2, pub, st_A) \leftarrow\$ A^{\underline{\mathsf{RO}}}(\,)$
  if $b = 0$: $(\alpha, \vec{\beta}) \leftarrow\$ E(key_1, key_2, pub, \mathrm{Rand})$
  $b' \leftarrow\$ A^{\underline{\mathsf{RO}}, \underline{\mathsf{Eval}}}(st_A)$; ret $b = b'$

$\underline{\mathsf{Eval}}(\vec{x})$:
  $(h_1, \vec{y}_1) \leftarrow\$ \mathsf{VIDPF.VEval}^{\underline{\mathsf{RO}}}(1, key_1, pub, \vec{x})$
  $(h_2, \vec{y}_2) \leftarrow\$ \mathsf{VIDPF.VEval}^{\underline{\mathsf{RO}}}(2, key_2, pub, \vec{x})$
  if $b = 0$ and $\mathsf{VIDPF.Verify}^{\underline{\mathsf{RO}}}(h_1, h_2) = 1$: ret $f_{\alpha,\vec{\beta}}(\vec{x})$
  else: ret $\vec{y}_1 + \vec{y}_2$

$\underline{\mathsf{RO}}(inp)$:
  if $\mathrm{Rand}[inp] = \bot$: $\mathrm{Rand}[inp] \leftarrow\$ \mathcal{Y}$
  ret $\mathrm{Rand}[inp]$
```

```
Game $\mathsf{Exp}^{\mathrm{priv}}_{\mathsf{VIDPF},S}(A)$:
  $b \leftarrow\$ \{0,1\}$; $(st_A, \alpha, \vec{\beta}, \hat{\jmath}) \leftarrow A(\,)$
  if $b = 0$: $(key_{\hat{\jmath}}, pub) \leftarrow\$ S_1(\hat{\jmath})$
  else: $(key_1, key_2, pub) \leftarrow\$ \mathsf{VIDPF.Gen}(\alpha, \vec{\beta})$
  $b' \leftarrow A^{\underline{\mathsf{Sketch}}}(st_A, key_{\hat{\jmath}}, pub)$; ret $b = b'$

$\underline{\mathsf{Sketch}}(\vec{x})$:
  if $b = 0$: $h \leftarrow S_2(\hat{\jmath}, key_{\hat{\jmath}}, pub, \vec{x})$
  else: $(h, \_) \leftarrow \mathsf{VIDPF.VEval}(3 - \hat{\jmath}, key_{3-\hat{\jmath}}, pub, \vec{x})$
  ret $h$
```

```
Game $\mathsf{Exp}^{\mathrm{priv}}_{\mathsf{DFLP},S}(A)$:
  $b \leftarrow\$ \{0,1\}$; $(\mathcal{X}, st_A) \leftarrow A()$
  if $b = 0$: $(st_S, jr, qr) \leftarrow S_1(|\mathcal{X}|)$
  else:
    $jr \leftarrow\$ \mathbb{F}^{jl}$; $qr \leftarrow\$ \mathbb{F}^{ql}$; $\mathit{\Delta} \leftarrow\$ \mathbb{F}^{el}$
    $\pi \leftarrow\$ \mathsf{DFLP.Prove}(\mathcal{X}, \mathit{\Delta}, jr)$
  $(x, st_A) \leftarrow A(st_A, jr, qr)$; assert $x \in \mathcal{X}$
  if $b = 0$: $\sigma \leftarrow S(st_S)$
  else: $\sigma \leftarrow \mathsf{DFLP.Query}(\mathsf{DFLP.Encode}(\mathit{\Delta}, x), \mathit{\Delta}, \pi, jr;\ qr)$
  $b' \leftarrow A(st_A, \sigma)$; ret $b = b'$
```

Figure 6: Games for defining extractability (top-left), and privacy (bottom-left) of VIDPFs and privacy of delayed-input FLP (right).

Definition 7. *A* **delayed-input FLP** DFLP *consists of the following algorithms:*

- $\mathsf{DFLP.Encode}(\mathit{\Delta} \in \mathbb{F}^{el}, x \in \mathbb{F}^n) \to e \in \mathbb{F}^{n'}$ *takes as input encoding randomness $\mathit{\Delta}$, and an input instance x. Returns an encoding of x; we let n' denote the length of the encoding. The function* $\mathsf{Encode}(\mathit{\Delta}, \cdot)$ *must be a linear function and invertible. We denote the inverse by* Decode.
- $\mathsf{DFLP.Prove}(\mathcal{X} \subseteq \mathbb{F}^n, \mathit{\Delta} \in \mathbb{F}^{el}, jr \in \mathbb{F}^{jl}) \to \pi \in \mathbb{F}^m$ *takes as input a* **set** *of possible instances, encoding randomness $\mathit{\Delta}$, and joint randomness jr. Produces output proof π.*
- $\mathsf{DFLP.Query}(e \in \mathbb{F}^{n'}, \mathit{\Delta} \in \mathbb{F}^{el}, \pi \in \mathbb{F}^m, jr \in \mathbb{F}^{jl}; qr \in \mathbb{F}^{ql}) \to \sigma \in \mathbb{F}^v$ *takes as input an encoded instance e, encoding randomness $\mathit{\Delta}$, proof π, joint randomness jr, and query randomness qr. Returns a verifier σ. The function* $\mathsf{Query}(\cdot, \cdot, \cdot, jr; qr)$ *must be linear.*
- $\mathsf{DFLP.Decide}(\sigma \in \mathbb{F}^v) \to acc \in \{0,1\}$*: Takes as input query responses σ and returns a boolean.*

If Prove *is restricted to sets $\mathcal{X}$ with $|\mathcal{X}| = k$ then we call the construction a* **delayed-k-input FLP**.

A delayed-input FLP should satisfy the following properties:

- **Completeness** (with respect to language $\mathcal{L}$): For all $\mathcal{X} \subseteq \mathcal{L}$, all $x \in \mathcal{X}$, and all $\mathit{\Delta}$:

$$\Pr[\mathsf{Decide}(\sigma) : jr \leftarrow\$ \mathbb{F}^{jl}; \pi \leftarrow\$ \mathsf{Prove}(\mathcal{X}, \mathit{\Delta}, jr);$$
$$\sigma \leftarrow\$ \mathsf{Query}(\mathsf{Encode}(\mathit{\Delta}, x), \mathit{\Delta}, \pi, jr)] = 1\,.$$

- **Soundness** (with respect to $\mathcal{L}$): The scheme should be sound in the usual sense of FLPs, with respect to the language $\mathcal{L}^* = \{(\mathsf{Encode}(\mathit{\Delta}, x), \mathit{\Delta}) \mid x \in \mathcal{L}\}$. In other words, it is hard for a malicious prover to generate a proof that verifies with respect to $(e, \mathit{\Delta}) \notin \mathcal{L}^*$.

```
Algorithm $\mathsf{Shard}(\alpha, n)$:
  // Construct the VIDPF key shares.
  $seed_1, seed_2 \leftarrow_{\$} \{0,1\}^\kappa$
  for $\ell \in [\eta]$:
    $\vec{\Delta}[\ell] \leftarrow \mathsf{RG}_2(seed_1, n \,\|\, \ell \,\|\, 1)$
          $+ \mathsf{RG}_2(seed_2, n \,\|\, \ell \,\|\, 2)$
    $\vec{\beta}[\ell] \leftarrow \mathsf{DFLP.Encode}(\vec{\Delta}[\ell], 1)$
  $(key_1, key_2, pub) \leftarrow_{\$} \mathsf{VIDPF.Gen}(\alpha, \vec{\beta})$
  // Prepare the joint randomness parts.
  $\vec{rseed}[1] \leftarrow \mathsf{RG}_5(seed_1, n \,\|\, 1 \,\|\, pub \,\|\, key_1)$
  $\vec{rseed}[2] \leftarrow \mathsf{RG}_5(seed_2, n \,\|\, 2 \,\|\, pub \,\|\, key_2)$
  // Generate the level proofs.
  for $\ell \in [\eta]$:
    $jseed \leftarrow \mathsf{RG}_6(0^\kappa, \ell \,\|\, \vec{rseed})$
    $jr \leftarrow \mathsf{RG}_1(jseed, n \,\|\, \ell)$
    $\pi \leftarrow_{\$} \mathsf{DFLP.Prove}(\{0,1\}, \vec{\Delta}[\ell], jr)$
    $\vec{pf}[\ell] \leftarrow \pi - \mathsf{RG}_3(seed_2, n \,\|\, \ell)$
  // Prepare the initial message and input shares.
  $x_1 \leftarrow (key_1, seed_1, \vec{pf})$
  $x_2 \leftarrow (key_2, seed_2)$
  $msg \leftarrow (pub, \vec{rseed})$
  ret $(msg, x_1, x_2)$

Algorithm $\mathsf{Unpack}(\hat{j}, x, n, \ell)$:
  if $\hat{j} = 1$: $(key, seed, \vec{pf}) \leftarrow x$; $\pi \leftarrow \vec{pf}[\ell]$
  else: $(key, seed) \leftarrow x$; $\pi \leftarrow \mathsf{RG}_3(seed, n \,\|\, \ell)$
  ret $(key, seed, \pi)$
```

```
Algorithm $\mathsf{Prep}(\hat{j}, sk, st, n, msg, x)$:
  if $st \in \mathcal{Q}_{\mathrm{Init}}$: // Process initial message from client
    $(\ell, \vec{pfx}) \leftarrow st$; $u \leftarrow |\vec{pfx}|$
    $(pub, \vec{rseed}) \leftarrow msg$; $(key, seed, \pi) \leftarrow \mathsf{Unpack}(\hat{j}, x, n, \ell)$
    $\Delta \leftarrow \mathsf{RG}_2(seed, n \,\|\, \ell \,\|\, \hat{j})$
    $\vec{rseed}[\hat{j}] \leftarrow \mathsf{RG}_5(seed, n \,\|\, \ell \,\|\, \hat{j} \,\|\, pub \,\|\, key)$
    $jseed \leftarrow \mathsf{RG}_6(0^\kappa, \vec{rseed})$
    $jr \leftarrow \mathsf{RG}_1(jseed, n \,\|\, \ell)$; $qr \leftarrow \mathsf{RG}_4(sk, n \,\|\, \ell)$
    $(h, \vec{y}) \leftarrow \mathsf{VIDPF.VEval}(\hat{j}, pub, key, \vec{pfx})$
    $inp \leftarrow \sum_{i \in [u]} \vec{y}[i]$
    $\sigma \leftarrow \mathsf{DFLP.Query}(inp, \Delta, \pi, jr;\ qr)$
    $msg \leftarrow (\sigma, \vec{rseed}[\hat{j}], h)$; $st \leftarrow (jseed, (\mathsf{DFLP.Decode}(\vec{y}[i]))_{i \in [u]})$
    ret $(\texttt{running}, st, msg)$
  // Process broadcast messages from aggregators
  $(jseed, \vec{y}) \leftarrow st$; $\Big((\sigma_1, rseed_1, h_1), (\sigma_2, rseed_2, h_2)\Big) \leftarrow msg$
  $acc \leftarrow \mathsf{DFLP.Decide}(\sigma_1 + \sigma_2)$
  if $acc$ and $jseed = \mathsf{RG}_6(0^\kappa, (rseed_1, rseed_2))$
      and $\mathsf{VIDPF.Verify}(h_1, h_2)$: ret $(\texttt{finished}, \vec{y})$
  else: ret $(\texttt{failed}, \bot)$

Algorithm $\mathsf{Agg}(\vec{y})$: ret $\sum_{i=1}^{|\vec{y}|} \vec{y}[i]$

Algorithm $\mathsf{Unshard}(\_, \vec{a})$: ret $\sum_{i=1}^{|\vec{a}|} \vec{a}[i]$

Algorithm $\mathsf{RG}_i(seed, cntxt)$:
  $l \leftarrow (jl, el, m, ql)$
  if $i \leq 4$: ret $\mathsf{Expand}[\mathsf{PRG}](seed, \texttt{label}_i \,\|\, cntxt, \mathbb{F}.p, l[i])$
  else: ret $\mathsf{PRG.Next}(\mathsf{PRG.Init}(seed, \texttt{label}_i \,\|\, cntxt), \kappa)$
```

Figure 7: Definition of 1-round, 2-party VDAF $\mathsf{Doplar}[\mathsf{VIDPF}, \mathsf{DFLP}, \mathsf{PRG}]$. Let $\texttt{label}_1, \ldots, \texttt{label}_6$ be arbitrary, distinct bitstrings.

- **Privacy:** In Figure 6 we define a game for delayed-input FLPs, in which the proof is generated using some set $\mathcal{X}$ of candidates, and later verified with respect to a particular $x \in \mathcal{X}$. A delayed-input FLP is δ-private if there exists a simulator S such that every A's advantage is $\mathsf{Adv}^{\mathrm{priv}}_{\mathsf{DFLP},S}(A) \leq \delta$, where

$$\mathsf{Adv}^{\mathrm{priv}}_{\mathsf{DFLP}}(A) = 2 \cdot \Pr[\mathsf{Exp}^{\mathrm{priv}}_{\mathsf{DFLP},S}(A)] - 1\,.$$

5.3 Construction

We specify our construction $\mathsf{Doplar}[\mathsf{VIDPF}, \mathsf{DFLP}, \mathsf{PRG}]$ in Figure 7. Its three components are: a verifiable IDPF VIDPF with input length η; a delayed-2-input FLP DFLP with input set $\{0,1\}$, proof length m, encoded input length n, encoding randomness length el, joint randomness length jl, and query randomness length ql; and a pseudorandom generator PRG (Section 2) with seed length κ. To be suitable for our construction, we must choose VIDPF and DFLP so that $\mathsf{VIDPF}.\mathbb{G}_\ell = \mathsf{DFLP}.\mathbb{F}^n$ for each $\ell \in [\eta]$.

To shard its measurement $\alpha \in \{0,1\}^\eta$, the client begins by running the VIDPF key generator on α. The initial state for Doplar encodes the "level" ℓ at which the VIDPF shares are to be evaluated; each candidate prefix must have length ℓ. (Recall from Section 2 that (V)IDPFs can be thought of as shares of values arranged in a binary tree with nodes labeled by prefixes.) For each level of the VIDPF tree, the client generates a delayed-input proof of the refined shares' validity; just as for Prio3 (Section 4), the joint randomness used at each level is derived from the aggregator's input shares. The VIDPF output is programmed so that the sum of the output shares corresponds to an encoded input for the delayed-input FLP.

To prepare a report for aggregation, the aggregators evaluate their VIDPF key shares at the desired candidate prefixes, then interact in order to check that (1) the joint randomness was computed correctly, (2) their refined shares are one-hot, and (3) the sum of their refined shares is either one or zero.

Allowed initial states. An initial state is valid if it consists of a sequence of candidate prefixes all having the same length. Moreover, each of the prefixes must be distinct. An initial state is allowed for $\mathsf{Doplar}[\mathsf{VIDPF}, \mathsf{DFLP}, \mathsf{PRG}]$ if the prefix length is distinct from all previous states for the same report. That is, the allowed-state algorithm $\mathsf{validSt}$ only permits a new state $st = (\ell, \vec{pfx})$ if ℓ is distinct for all previous states and each of the prefixes $\vec{pfx}$ is distinct.

Remark 6. *Although not addressed in Boneh et al. [18] explicitly, this restriction on the candidate prefixes is necessary for Poplar as well, as re-using the correlated randomness shared by the client would reveal information about the secret-shared vector.*

Consistency. The set of refined measurements for Doplar are one-hot vectors over the field $\mathbb{F}$ for which the non-zero element is equal to 0 or 1. For a given initial state $(\ell, \vec{pfx})$, this can be computed from the VIDPF public share and key shares by evaluating the shares on each of the prefixes $\vec{pfx}$. Since the VIDPF is a point function and the prefixes are distinct, the vector of VIDPF outputs will contain at most one nonzero entry. Aggregation consistency for Doplar is similarly straight-forward, since the refined share space and aggregate share space are the same and both aggregation and unsharding are vector summation. When we let $\mathsf{finishResult}$ be vector summation as well, the desired property is trivially true.

Security. Let $\Pi = \mathsf{Doplar}[\mathsf{VIDPF}, \mathsf{DFLP}, \mathsf{PRG}]$ as specified above. Let $\mathcal{N}$ be the nonce space and let κ be the seed length for PRG.

Theorem 3. *Modeling each RG_i in Figure 7 as a random oracle, if DFLP is ϵ-sound, then for all t_A-time adversaries A and t_E-time extractors E there exists a $O(t_A + q_{\mathsf{Prep}} t_E)$-time adversary B for which*

$$\begin{aligned}\mathsf{Adv}^{\mathrm{robust}}_{\Pi}(A) \leq 2(q_{\mathsf{RG}} + q_{\mathsf{Prep}}) \cdot \epsilon + \frac{(q_{\mathsf{RG}} + 3q_{\mathsf{Prep}})^2}{2^\kappa} \\ + q_{\mathsf{Prep}} \cdot \mathsf{Adv}^{\mathrm{extract}}_{\mathsf{VIDPF},E}(B)\,,\end{aligned}$$

where A makes q_{Prep} queries to $\underline{\mathsf{Prep}}$ and a total of q_{RG} queries to its random oracles.

Proof sketch. The proof has a similar structure to Theorem 1 in that the last step is a reduction to the soundness of DFLP. However in order to use this, we must first revise the game so that the challenge input issued by the malicious prover P^* was constructed from the sum of refined shares that are otherwise valid (i.e., one-hot). Using the extractability property of VIDPF, we can simplify the winning condition by extracting the the input measurement from the adversary's random oracle queries and use it to compute the refined measurement whenever the one-hotness check succeeds. Refer to Appendix C.3 for the proof. □

Theorem 4. *For all t_A-time adversaries A and t'-time simulators S, T there exist $O(t_A + q_{\mathsf{Shard}} t')$-time adversaries B, C for which*

$$\begin{aligned}\mathsf{Adv}^{\mathrm{priv}}_{\Pi,1}(A) \leq 2q_{\mathsf{Shard}} \Big(\mathsf{Adv}^{\mathrm{priv}}_{\mathsf{VIDPF},S}(B) + \eta \cdot \mathsf{Adv}^{\mathrm{priv}}_{\mathsf{DFLP},T}(C) \\ + \frac{\eta q_{\mathsf{RG}} + q_{\mathsf{Shard}}}{|\mathcal{N}|} + \frac{3q_{\mathsf{RG}}}{2^{\kappa-1}} \Big)\,,\end{aligned}$$

where each RG_i in Figure 7 is modeled as a random oracle, adversary A makes a total of q_{RG} queries to all of its random oracles and q_{Shard} queries to $\underline{\mathsf{Shard}}$.

Proof sketch. The reduction to DFLP privacy follows the same lines as Theorem 2 except there are $\eta \cdot q_{\mathsf{Shard}}$ different hybrid worlds in the last step. Privacy of VIDPF is used to ensure that the simulation of the boundary world can be carried out without access to the input measurement. Refer to Appendix C.4 for the proof. □

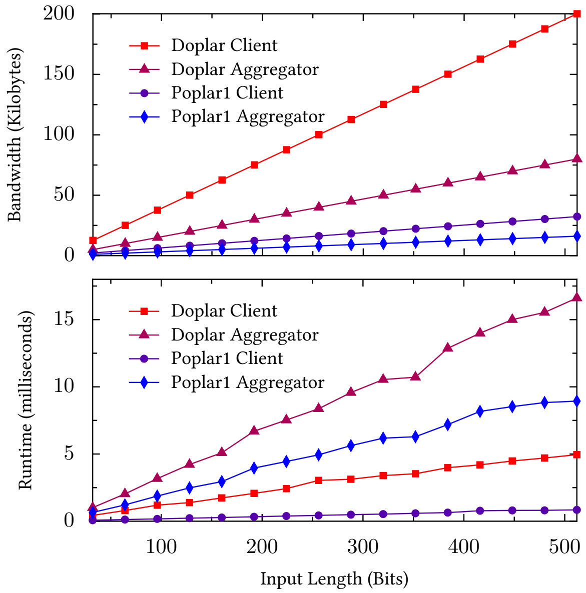

Figure 8: Bandwidth (top) and runtime (bottom) for Doplar and Poplar1.

5.4 Performance Evaluation

In this section we compare the cost of Doplar to Poplar1 in terms of communication (total bits written to the wire) and computation. The parameters chosen for Poplar1 by the specification [9] match those in the performance evaluation conducted by Boneh et al. [17]. We therefore take these parameters as our basis for comparison. In the following, we have instantiated VIDPF and DFLP as described in Appendix A and Appendix B respectively.

Boneh et al. [17] claim a per-report robustness bound of roughly $2/|\mathbb{F}|$, where $\mathbb{F}$ is the field chosen for the inner nodes.[5] They choose a 62-bit field. In order to obtain the same robustness bound, while permitting the adversary at most 2^{64} queries to its random oracles, we need to use a 128-bit field for Doplar. For both constructions, we instantiate the PRG with AES-128 as described in [9, Section 6.2] (hence the seed length is $\kappa = 128$).

Communication overhead. In Figure 8 we plot the communication cost of Doplar and Poplar1 for various choices of the input length η. We plot the total number of kilobytes sent by each client. We also plot the total number of kilobytes sent by each aggregator, per report, over all η rounds of aggregation. As one would expect, the communication cost for Doplar scales linearly with the input length. However, the client's bandwidth is about 6 times that of Poplar1; and the Aggregator's bandwidth is about 5 times.

Computational overhead. To evaluate Doplar's computational overhead, we implement a prototype[6] and benchmark it against an existing implementation of Poplar1. The ISRG (Internet Security Research Group) maintains Rust implementations of the current crop of VDAF standard candidates.[7] The code includes a work-in-progress version of Poplar1 (on a development branch, as of this writing) as well as the FLP and IDPF primitives we use in our own implementation of Doplar.

We use the Criterion framework for Rust.[8] All benchmarks reported below were run on a 2019 MacBook Pro (2.6 GHz 6-Core Intel Core i7) running rustc version 1.67.1 and cargo-criterion version 1.1.0. The default parameters were used, except the measurement time was set to 30 seconds for all benchmarks.

Microbenchmarks for sharding. To benchmark the client, we chose a random input string of the desired length, then measured the runtime of the sharding algorithm on that input. Figure 8 shows the runtimes for lengths ranging from 32 to 512 bits. From these data we see that sharding is about 6 times as expensive

[5] Poplar1 uses a smaller field for the inner nodes of the IDPF tree than the leaf nodes.

[6] `https://github.com/cloudflareresearch/doplar/tree/cjpatton/PoPETS-2023.4-Artifact`

[7] Source code for the `prio` crate: `https://github.com/divviup/libprio-rs`

[8] Criterion: `https://docs.rs/criterion/latest/criterion/`

for Doplar as for Poplar1. However, sharding a 512-bit input takes only 5 milliseconds, which is still quite practical. (Moreover, there is more room for optimization of our prototype.)

Microbenchmarks for preparation. Due to the highly parallelizable nature of VDAFs, much of the time the aggregators spend on executing the protocol is network-bound. However, it is useful to assess the amount of CPU time spent on processing a single report. To do so, we report microbenchmarks for per-report preparation, specifically how much time it takes an aggregator to compute its (first) broadcast message from the initial state provided by the collector and the input share provided by the client. Let us call this "preparation initialization".

One complicating factor is that the runtime of IDPF evaluation depends intrinsically on the distribution of the batch of measurements and the heavy-hitters threshold used. (We refer the reader to Algorithm 3 in Boneh et al. [18] for details.) To address this, we generated a synthetic batch of measurements and computed the prefix tree (cf. [18, Section 5.1]) for the desired threshold, then ran preparation initialization on the longest paths of this tree.[9]

The following experiment was run 10 times. Following Boneh et al. [18], we sample random input strings from a Zipf distribution (with parameter 1.03 and support 128), then compute the prefix tree with a heavy-hitters threshold of 10. We chose a batch size of 1000. For both Doplar and Poplar, run Criterion to measure the runtime of preparation for the longest paths of the tree.

Figure 8 shows the runtime averaged over all trials for lengths ranging from 32 to 512 bits. From these data we see that preparation is only about 1.75 times as expensive for Doplar as for Poplar1. This is not surprising, given that the runtime is dominated by IDPF evaluation, which in turn depends on the number of candidates.

Level skipping. One way to improve bandwidth for both schemes is to "skip" IDPF evaluation at certain levels. For example, if we descend the IDPF tree in τ-bit increments instead of 1-bit increments, then (1) our VIDPF construction requires one-hot check material only in every τ-th level, and (2) the Doplar construction requires DFLPs only at every τ-th level.[10] As a result, these major contributors to communication cost are reduced by a factor of τ. Additionally, the process of aggregating (traversing the tree of prefixes to find heavy hitters) requires fewer rounds by a factor of τ. The trade-off is that we consider more candidate prefixes at each level—i.e., at each step we consider the 2^τ descendants at depth τ from each candidate—but this cost is amortized over the batch.

Notably, the impact of this optimization is more significant for Doplar than for Poplar1. (For example, a "skip factor" of $\tau = 2$, i.e., skipping every other level, reduces the client's overhead from 6 to 5 times that of Poplar1 with the same optimization.) This is primarily due to the reduction in the number of delayed-input proofs, which make up the bulk of the first input share. (The second input share compresses its shares of the proofs into a single PRG seed.)

6 Conclusion and Future Work

The PPM working group's ambition is to preserve user privacy even as software systems rely increasingly on gaining insights into user behavior. Our work aims to help ensure that this effort rests on firm formal foundations. However, we leave open a number of directions for future work. We discuss two in the remainder.

Security analysis of DAP. The definitions in this paper apply to VDAFs, which are only a component of the DAP specification [30]. Thus, our work necessarily leaves open the security of the end-to-end protocol. There are two important questions. First, DAP is designed to inherit the security properties of VDAF, i.e., one would hope that whatever can be proven about the VDAF also holds when the VDAF is instantiated in the real-world environment in which DAP runs. One way to address this is to formulate the problem in terms of *indifferentiability* [49]: if DAP's execution can be shown to be indifferentiable from the execution of the VDAF in the idealized environment described here, then any attack against DAP can be translated into an attack against the underlying VDAF.

The other important question is whether DAP meets its own security goals, which, depending on the application, might go beyond what can be achieved with a VDAF alone. Consider that whether MPC-style

[9] Note that IDPFs can be implemented with cross-aggregation cache, which amortizes longest-path evaluation over multiple aggregations.

[10] The underlying (non-verifiable) IDPF is still organized as a binary tree, so its cost is not affected.

definitions like ours are enough for privacy depends intrinsically on the nature of the measurements being collected and how they are aggregated. It is one thing to ensure that we securely compute the aggregate; it is another to ensure that the aggregate itself does not leak "too much" information about the measurements. In particular, in many applications it will be useful to achieve differential privacy (DP) [28] in addition to secure computation. There are definitions of DP that extend to the multi-party setting [45, 52], and a number of works have considered MPC protocols for aggregation functionalities that also guarantee differential privacy of the outputs [51, 36, 10]. We hope to see future work extend this investigation to specific VDAFs.

Doplar improvements. For some applications, it would be useful for Doplar (or Poplar1) if the leaf output could be "weighted", i.e., a number in range $\{a, \ldots, b\}$ rather than $\{0, 1\}$. (Consider the ad-conversion use case from Section 1: it might be useful to know not only how many purchases were made per ad impression, but the total amount of money that was spent.) The delayed-k-input FLP paradigm may allow for this generalization, if schemes can be constructed for $k > 2$. (In this work, we only construct the delayed-2-input FLP needed for plain heavy hitters.)

There is also room for improvement of the communication cost. Despite the round reduction, the higher bandwidth may be prohibitive for some applications. However, we are optimistic that the bandwidth can be improved. Future work should focus on the delayed-2-input FLP. The current instantiation (Appendix B), while simple, effectively doubles the proof size of the base FLP.

Acknowledgements

Thank you to the anonymous reviewers from the PETS 2023 program committee whose feedback helped us improve a number of technical aspects of our paper. Thank you as well to Christopher Wood who helped us position this work in the context of the ongoing standardization effort at IETF. Finally, thanks to Nikita Borisov, Sofía Celi, Tanya Verma, Tara Whalen, and Avani Wildani for editorial improvements.

Hannah and Mike carried out their work on this paper while visiting Cloudflare Research. This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

[1] Privacy preserving measurement (2022), URL `https://datatracker.ietf.org/wg/ppm/about/`

[2] Private Advertising Tecnology Community Group (2022), URL `https://www.w3.org/community/patcg/`

[3] Abdalla, M., Haase, B., Hesse, J.: Security analysis of cpace. Cryptology ePrint Archive, Paper 2021/114 (2021), URL `https://eprint.iacr.org/2021/114`

[4] Abdalla, M., Haase, B., Hesse, J.: CPace, a balanced composable PAKE. Internet-Draft draft-irtf-cfrg-cpace-06, Internet Engineering Task Force (Jul 2022), URL `https://datatracker.ietf.org/doc/draft-irtf-cfrg-cpace/06/`, work in Progress

[5] Addanki, S., Garbe, K., Jaffe, E., Ostrovsky, R., Polychroniadou, A.: Prio+: Privacy preserving aggregate statistics via boolean shares. Cryptology ePrint Archive, Report 2021/576 (2021), `https://ia.cr/2021/576`

[6] Anderson, E., Chase, M., Durak, F.B., Ghosh, E., Laine, K., Weng, C.: Aggregate measurement via oblivious shuffling. Cryptology ePrint Archive, Report 2021/1490 (2021), `https://ia.cr/2021/1490`

[7] Apple, Google: Exposure Notification Privacy-preserving Analytics (ENPA). White paper (2021), `https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf`

[8] Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. J. ACM **45**(3), 501–555 (may 1998), ISSN 0004-5411, doi:10.1145/278298.278306, URL `https://doi.org/10.1145/278298.278306`

[9] Barnes, R., Patton, C., Schoppmann, P.: Verifiable Distributed Aggregation Functions. Internet-Draft draft-irtf-cfrg-vdaf-03, Internet Engineering Task Force (Aug 2022), URL `https://datatracker.ietf.org/doc/draft-irtf-cfrg-vdaf/03/`, work in Progress

[10] Bell, J., Gascón, A., Ghazi, B., Kumar, R., Manurangsi, P., Raykova, M., Schoppmann, P.: Distributed, private, sparse histograms in the two-server model. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 307–321 (2022)

[11] Bell, J., Gascón, A., Lepoint, T., Li, B., Meiklejohn, S., Raykova, M., Yun, C.: Acorn: Input validation for secure aggregation. Cryptology ePrint Archive (2022), URL `https://eprint.iacr.org/2022/1461`

[12] Bell, J.H., Bonawitz, K.A., Gascón, A., Lepoint, T., Raykova, M.: Secure single-server aggregation with (poly) logarithmic overhead. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 1253–1269 (2020)

[13] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security, pp. 62–73, CCS '93, ACM, New York, NY, USA (1993), ISBN 0-89791-629-8

[14] Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) Advances in Cryptology - EUROCRYPT 2006, pp. 409–426, Springer Berlin Heidelberg, Berlin, Heidelberg (2006), ISBN 978-3-540-34547-3

[15] Bitansky, N., Chiesa, A., Ishai, Y., Paneth, O., Ostrovsky, R.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) Theory of Cryptography, pp. 315–333, Springer Berlin Heidelberg, Berlin, Heidelberg (2013), ISBN 978-3-642-36594-2

[16] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1175–1191 (2017)

[17] Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear pcps. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019, pp. 67–97, Springer International Publishing, Cham (2019), ISBN 978-3-030-26954-8

[18] Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Lightweight techniques for private heavy hitters. In: IEEE Symposium on Security and Privacy, pp. 762–776, IEEE (2021)

[19] Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1292–1303 (2016)

[20] Brickell, J., Shmatikov, V.: Efficient anonymity-preserving data collection. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 76–85 (2006)

[21] Canetti, R.: Universally composable security. J. ACM **67**(5) (sep 2020), ISSN 0004-5411, doi:10.1145/3402457, URL `https://doi.org/10.1145/3402457`

[22] de Castro, L., Polychroniadou, A.: Lightweight, maliciously secure verifiable function secret sharing. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I, Lecture Notes in Computer Science, vol. 13275, pp. 150–179, Springer (2022), doi:10.1007/978-3-031-06944-4_6, URL `https://doi.org/10.1007/978-3-031-06944-4_6`

[23] Corrigan-Gibbs, H., Boneh, D.: Prio: Private, robust, and scalable computation of aggregate statistics. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pp. 259–282, USENIX Association, Boston, MA (Mar 2017), ISBN 978-1-931971-37-9, URL `https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs`

[24] Danezis, G., Fournet, C., Kohlweiss, M., Zanella-Béguelin, S.: Smart meter aggregation via secret-sharing. In: Proceedings of the first ACM workshop on Smart energy grid security, pp. 75–80 (2013)

[25] Davidson, A., Snyder, P., Quirk, E., Genereux, J., Livshits, B., Haddadi, H.: Star: Secret sharing for private threshold aggregation reporting. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 697–710 (2022)

[26] Davis, H., Patton, C., Rosulek, M., Schoppmann, P.: Verifiable distributed aggregation functions. Cryptology ePrint Archive, Paper 2023/130 (2023), URL `https://eprint.iacr.org/2023/130`

[27] Duan, Y., Canny, J., Zhan, J.: {P4P}: Practical {Large-Scale}{Privacy-Preserving} distributed computation robust against malicious users. In: 19th USENIX Security Symposium (USENIX Security 10) (2010)

[28] Dwork, C.: Differential privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) Automata, Languages and Programming, pp. 1–12, Springer Berlin Heidelberg, Berlin, Heidelberg (2006), ISBN 978-3-540-35908-1

[29] Elahi, T., Danezis, G., Goldberg, I.: Privex: Private collection of traffic statistics for anonymous communication networks. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1068–1079 (2014)

[30] Geoghegan, T., Patton, C., Rescorla, E., Wood, C.A.: Distributed Aggregation Protocol for Privacy Preserving Measurement. Internet-Draft draft-ietf-ppm-dap-02, Internet Engineering Task Force (Sep 2022), URL `https://datatracker.ietf.org/doc/draft-ietf-ppm-dap/02/`, work in Progress

[31] Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology – EUROCRYPT 2014, pp. 640–658, Springer Berlin Heidelberg, Berlin, Heidelberg (2014), ISBN 978-3-642-55220-5

[32] Green, M., Ladd, W., Miers, I.: A protocol for privately reporting ad impressions at scale. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, p. 1591–1601, Association for Computing Machinery, New York, NY, USA (2016), ISBN 9781450341394, doi:10.1145/2976749.2978407, URL `https://doi.org/10.1145/2976749.2978407`

[33] Guo, C., Katz, J., Wang, X., Yu, Y.: Efficient and secure multiparty computation from fixed-key block ciphers. In: 2020 IEEE Symposium on Security and Privacy (SP), pp. 825–841, IEEE (2020)

[34] Guo, X., Yang, K., Wang, X., Zhang, W., Xie, X., Zhang, J., Liu, Z.: Half-tree: Halving the cost of tree expansion in cot and dpf. Cryptology ePrint Archive, Paper 2022/1431 (2022), URL `https://eprint.iacr.org/2022/1431`

[35] Hohenberger, S., Myers, S., Pass, R., et al.: Anonize: A large-scale anonymous survey system. In: 2014 IEEE Symposium on Security and Privacy, pp. 375–389, IEEE (2014)

[36] Humphries, T., Akhavan Mahdavi, R., Veitch, S., Kerschbaum, F.: Selective mpc: Distributed computation of differentially private key-value statistics. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 1459–1472 (2022)

[37] III, J.J.P., Charles, D., Gilton, D., Jung, Y.H., Parsana, M., Anderson, E.: Masked lark: Masked learning, aggregation and reporting workflow (2021)

[38] Ishai, Y., Kushilevitz, E., Ostrovsky, R.: Efficient arguments without short pcps. In: Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07), pp. 278–291 (2007), doi:10.1109/CCC.2007.10

[39] Jangir, P., Koti, N., Kukkala, V.B., Patra, A., Gopal, B.R., Sangal, S.: Vogue: Faster computation of private heavy hitters. Cryptology ePrint Archive, Paper 2022/1561 (2022), URL `https://eprint.iacr.org/2022/1561`

[40] Jawurek, M., Kerschbaum, F.: Fault-tolerant privacy-preserving statistics. In: International Symposium on Privacy Enhancing Technologies Symposium, pp. 221–238, Springer (2012)

[41] Katz, J., Ostrovsky, R.: Round-optimal secure two-party computation. In: Annual International Cryptology Conference, pp. 335–354, Springer (2004)

[42] Kursawe, K., Danezis, G., Kohlweiss, M.: Privacy-friendly aggregation for the smart-grid. In: International Symposium on Privacy Enhancing Technologies Symposium, pp. 175–191, Springer (2011)

[43] Lapidot, D., Shamir, A.: Publicly verifiable non-interactive zero-knowledge proofs. In: Menezes, A., Vanstone, S.A. (eds.) Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings, Lecture Notes in Computer Science, vol. 537, pp. 353–365, Springer (1990), doi:10.1007/3-540-38424-3_26, URL https://doi.org/10.1007/3-540-38424-3_26

[44] Melis, L., Danezis, G., De Cristofaro, E.: Efficient private statistics with succinct sketches. arXiv preprint arXiv:1508.06110 (2015)

[45] Mironov, I., Pandey, O., Reingold, O., Vadhan, S.: Computational differential privacy. In: Annual International Cryptology Conference, pp. 126–142, Springer (2009)

[46] Molteni, D.: Improving the WAF with machine learning. Cloudflare blog (2022), URL https://blog.cloudflare.com/waf-ml/

[47] Mouris, D., Sarkar, P., Tsoutsos, N.G.: PLASMA: Private, lightweight aggregated statistics against malicious adversaries with full security. Cryptology ePrint Archive, Paper 2023/080 (2023), URL https://eprint.iacr.org/2023/080, https://eprint.iacr.org/2023/080

[48] Mozilla: Origin Telemetry (2022), URL https://firefox-source-docs.mozilla.org/toolkit/components/telemetry/collection/origin.html

[49] Patton, C., Shrimpton, T.: Quantifying the security cost of migrating protocols to practice. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, pp. 94–124, Springer International Publishing, Cham (2020), ISBN 978-3-030-56784-2

[50] Popa, R.A., Blumberg, A.J., Balakrishnan, H., Li, F.H.: Privacy and accountability for location-based aggregate statistics. In: Proceedings of the 18th ACM conference on Computer and communications security, pp. 653–666 (2011)

[51] Roth, E., Noble, D., Falk, B.H., Haeberlen, A.: Honeycrisp: Large-scale differentially private aggregation without a trusted core. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles, p. 196–210, SOSP '19, Association for Computing Machinery, New York, NY, USA (2019), ISBN 9781450368735, doi:10.1145/3341301.3359660, URL https://doi.org/10.1145/3341301.3359660

[52] Schoppmann, P., Vogelsang, L., Gascón, A., Balle, B.: Secure and scalable document similarity on distributed databases: Differential privacy to the rescue. Proceedings on Privacy Enhancing Technologies **2**, 209–229 (2020)

[53] Taubeneck, E., Thomson, M., Savage, B., Case, B., Masny, D., Jain, R.: Ipa end to end protocol. Proposal submitted to the PATCG working group of the W3 (2022), URL https://github.com/patcg-individual-drafts/ipa/blob/main/IPA-End-to-End.md

A Instantiating VIDPF

In this section we present our proposed VIDPF construction.

De Castro-Polychroniadou technique. De Castro & Polychroniadou [22] (hereafter DP22) proposed the following simple and elegant technique to verify that a vector is one-hot. Consider a vector $\vec{v}$ that is additively secret-shared $\vec{v} = \vec{v}_1 \oplus \vec{v}_2$. For simplicity, we describe the technique assuming that the sharing is with respect to XOR, since in that case the shares of zero are *identical strings.* The technique adapts readily to the more general case of additive shares over any group. Assume also that the parties have additive shares of a *binary* indicator vector $\vec{b} = \vec{b}_1 \oplus \vec{b}_2$, which is nonzero exactly in the same positions that $\vec{v}$ is.

First, observe that the parties can easily verify whether they hold shares of an all-zeroes vector, since this happens if and only if their shares (as strings) are identical. They can simply exchange and compare hashes of their share-vectors (although see our remark below for a disclaimer about this idea). The technique of DP22 is to adjust a one-hot vector into an all-zeroes vector, with the help of the dealer.

Define

$$\mathsf{adjust}(\vec{v}_i, \vec{b}_i, C) = \Big(H(1, \vec{v}_i[1]) \oplus \vec{b}_i[1] \cdot C, \quad H(2, \vec{v}_i[2]) \oplus \vec{b}_i[2] \cdot C, \ldots \Big)$$

If $\vec{v}$ and $\vec{b}$ are nonzero in (only) position i^*, then set $C^* = H(i^*, \vec{v}_1[i^*]) \oplus H(i^*, \vec{v}_2[i^*])$. Now consider the result of both shareholders applying $\mathsf{adjust}(\cdot, \cdot, C^*)$ to their shares:

- In positions $i \neq i^*$ where they share zero, we have $\vec{v}_1[i] = \vec{v}_2[i]$ and $\vec{b}_1[i] = \vec{b}_2[i]$. For these positions in the output of adjust, both parties will compute identical strings.
- In position i^*, the parties have $\vec{b}_1[i^*] \neq \vec{b}_2[i^*]$. By symmetry, suppose $\vec{b}_1[i^*] = 1$ and $\vec{b}_2[i^*] = 0$. Then the first party will compute

  $$\begin{aligned} &H(i^*, \vec{v}_1[i^*]) \oplus C^* \\ &\quad = H(i^*, \vec{v}_1[i^*]) \oplus \big(H(i^*, \vec{v}_1[i^*]) \oplus H(i^*, \vec{v}_2[i^*]) \big) \\ &\quad = H(i^*, \vec{v}_2[i^*]) \end{aligned}$$

  and the second party will compute $H(i^*, \vec{v}_2[i^*])$ as well.

In all cases, both parties will compute the same output of adjust, which they can check for equality by exchanging and comparing hashes. Hence, the dealer will compute the C^* value and include it in the parties' DPF keys. They can use C^* to perform their verification.

To see why the DP22 approach is sound, suppose the parties hold shares of a non-one-hot vector — i.e., it is nonzero at positions $i \neq i'$. Do both parties compute the same output of adjust? This can only happen if C^* value somehow corrects both positions i and i', and this happens only when

$$\begin{aligned} & H(i, \vec{v}_1[i]) \oplus H(i, \vec{v}_2[i]) = C^* = H(i', \vec{v}_1[i']) \oplus H(i', \vec{v}_2[i']) \\ \iff & H(i, \vec{v}_1[i]) \oplus H(i, \vec{v}_2[i]) \oplus H(i', \vec{v}_1[i']) \oplus H(i', \vec{v}_2[i']) = 0 \end{aligned}$$

The construction is therefore sound if it is hard to find "multi-collisions" of this form in H. In particular, if H is a random oracle with output length 4κ then an adversary making $q < 2^\kappa$ queries to H can find such a collision with probability bounded by $q^4/2^{4\kappa} \ll q/2^\kappa$.

Regarding privacy, there is one subtle issue that must be considered. Suppose party #1 holds its share $\vec{v}_1$ and the correction value $C^* = H(i^*, \vec{v}_1[i^*]) \oplus H(i^*, \vec{v}_2[i^*])$. Suppose this party has a guess for i^* and a guess for the nonzero value $v = \vec{v}_1[i^*] \oplus \vec{v}_2[i^*]$. Then she can verify this guess by checking whether $C^* = H(i^*, \vec{v}_1[i^*]) \oplus H(i^*, \vec{v}_1[i^*] \oplus v)$ — all values she knows. Hence, C^* exposes an offline dictionary attack on the secret values i^* and $\vec{v}[i^*]$. If $\vec{v}[i^*]$ is high entropy, then this is no vulnerability at all. But if $\vec{v}[i^*]$ is known to be a small value like 1 (as is the case in many applications), then this issue allows a corrupt shareholder to unilaterally learn i^*, violating privacy. We resolve this by simply ensuring that the dealer encodes a random element at the one-hot position (in addition to a potentially low-entropy desired value).[11]

Extending to incremental DPF. The technique of DP22 is well-suited for DPFs. In an incremental DPF (IDPF), we can apply their technique to each prefix-length. However, this guarantees only that each

[11] We have chosen to describe our VIDPF to use an underlying IDPF as a black-box. When this is the case, we must ensure that the IDPF outputs have sufficient entropy for the one-hotness check. If we were to instead to analyze our VIDPF (instantiated with a natural IDPF construction) as a *monolithic construction*, it is likely that the underlying IDPF would already have internal entropy available that could be used for the one-hotness check. I.e., we may be able to obtain smaller share sizes by exploiting internal properties of the underlying IDPF.

```
$\mathsf{VIDPF.Gen}(\alpha \in \{0,1\}^\eta, \vec{\beta} \in \mathbb{G}_1 \times \cdots \times \mathbb{G}_\eta)$:
for $\ell \in [\eta]$:
    $\vec{R}[\ell] \leftarrow\!\$\ \{0,1\}^\kappa$
    $\vec{\beta}^*[\ell] \leftarrow (1, \vec{\beta}[\ell], \vec{R}[\ell])$
$(key_1, key_2, pub) \leftarrow \mathsf{IDPF.Gen}(\alpha, \vec{\beta}^*)$
for $\ell \in [\eta]$:
    $pfx \leftarrow \alpha[1:\ell]$
    $(\_, data_1, R_1) \leftarrow \mathsf{IDPF.Eval}(1, key_1, pub, pfx)$
    $(\_, data_2, R_2) \leftarrow \mathsf{IDPF.Eval}(2, key_2, pub, pfx)$
    $\vec{C}[\ell] \leftarrow \mathsf{RG}(pfx \,\|\, -data_1 \,\|\, -R_1)$
            $\oplus\ \mathsf{RG}(pfx \,\|\, data_2 \,\|\, R_2)$
$pub^* \leftarrow (pub, \vec{C})$
ret $(key_1, key_2, pub^*)$

$\mathsf{VIDPF.VEval}(id, key, pub^*, \vec{x})$:
$(pub, \vec{C}) \leftarrow pub^*$
for $i \in [\ |\vec{x}|\ ]$
    $\vec{y}[i] \leftarrow \mathsf{IDPF.Eval}(id, key, pub, \vec{x}[i])$
    $(b, \vec{data}[i], R) \leftarrow \vec{y}[i]$
    $h \leftarrow h \,\|\, \mathsf{adjust}(id, key, pub^*, b, \vec{x}[i])$
ret $(h, \vec{data})$

$\mathsf{VIDPF.adjust}(id, key, pub^*, b, x)$: // a helper procedure
$(pub, \vec{C}) \leftarrow pub^*$
if $|x| = 0$: ret $x$ // length of $x$ as a bit string
$(\_, d, R) \leftarrow \mathsf{IDPF.Eval}(id, key, pub, x)$
$prefix \leftarrow \mathsf{adjust}(id, key, pub^*, b, x[\,1:|x|-1\,])$
ret $prefix \,\|\, \Big(\mathsf{RG}(x \,\|\, (-1)^{id} d \,\|\, (-1)^{id} R) \oplus b \cdot \vec{C}[|x|]\Big)$

$\mathsf{VIDPF.Verify}(h_1, h_2)$:
    ret $h_1 == h_2$
```

Figure 9: VIDPF construction $\mathsf{VIDPF[IDPF]}$, based on any IDPF. If the VIDPF is to be instantiated with groups $\mathbb{G}_1, \ldots, \mathbb{G}_\eta$ then the underlying IPDF is instantiated with groups $\widetilde{\mathbb{G}}_1, \ldots, \widetilde{\mathbb{G}}_\eta$, where $\widetilde{\mathbb{G}}_\ell = \{0,1\} \times \mathbb{G}_\ell \times \{0,1\}^\kappa$.

prefix-length corresponds to some point function. It does not necessarily guarantee that the point functions of the different prefix-lengths satisfy the prefix condition that is needed in an IDPF.

In our construction, we extend the DP22 technique to IDPFs. For each evaluation of the IDPF — say, at point x — we compute the adjustment strings using the DP22 technique, for *x and all of its prefixes.* This alone is not enough to guarantee the prefix property. To "tie different prefix lengths together," we ask the shareholders to compute the adjustment strings with respect to the *same sharings of the indicator bit*, for all the prefixes of x. We show that this forces the point functions at every prefix-length to be prefix-consistent.

Immediate Optimizations in an Implementation. Our construction evaluates the underlying IDPF on all prefixes of the given strings. Doing this naïvely would increase the computational costs by a factor of ℓ when evaluating on strings of length ℓ. However, these extra evaluations are essentially free in existing IDPFs — while evaluating at string x, these constructions already evaluate all prefixes of x along the way. A reasonable implementation of our VIDPF will take advantage of this fact.

The verification value h produced by VEval is a very long string, consisting of $\ell \cdot 4\kappa$ bits for each query point of length ℓ. If parties are to exchange these h values in an application of our VIDPF, it would account for a significant fraction of the total communication. However, the Verify algorithm that uses these h values merely checks them for equality. Therefore, it suffices for each party to send only a collision-resistant hash of their h value, which can have fixed length only 2κ. This optimization changes the concrete security bound for VIDPF soundness, by adding a term for the probability of finding a collision under the hash function.

Lemma 1. *Let* IDPF *be an IDPF and* RG *be a random oracle with outputs of length* 4κ*. Let A be an adversary making q queries to* RG*. There is a $O(t_A)$-time adversary A' such that the construction* $\mathsf{VIDPF[IDPF]}$ *in Figure 9 satisfies the following:*

$$\mathsf{Adv}^{\mathrm{extract}}_{\mathsf{VIDPF[IDPF]},E}(A) \le (q^4 + q^2)/2^{4\kappa}$$
$$\mathsf{Adv}^{\mathrm{priv}}_{\mathsf{VIDPF[IDPF]}}(A) \le \mathsf{Adv}^{\mathrm{priv}}_{\mathsf{IDPF}}(A') + q/2^\kappa$$

Proof. Correctness of our construction follows from the discussion above, and is the same as in DP22.

Extractability: We begin with a few observations, which hold for all VIDPF keys, even adversarially generated ones:

```
$E(key_1, key_2, pub^*, \mathrm{Rand})$:
$(pub, \vec{C}) \leftarrow pub^*$
if $\mathcal{E}_1$ or $\mathcal{E}_2$:  // defined in the text, here with respect to oracle queries listed in Rand
  abort
$\alpha \leftarrow$ empty string
for $\ell \in [\eta]$:
  if $\exists a \in \{0,1\}, y_1, R_1, y_2, R_2$ such that
      $\vec{C}[\ell] = \mathrm{Rand}[(\alpha \| a) \,\|\, -y_1 \,\|\, -R_1] \oplus \mathrm{Rand}[(\alpha \| a) \,\|\, y_1 \,\|\, R_2]$
    $\alpha \leftarrow \alpha \,\|\, a$
    $\vec{\beta}[\ell] \leftarrow y_1 + y_2$
  else: $\alpha \leftarrow \alpha \,\|\, 0$; $\vec{\beta}[\ell] \leftarrow 0$
ret $(\alpha, \vec{\beta})$
```

Figure 10: Extractor for the proof of Lemma 1.

Observation: If $\mathsf{adjust}(1, key_1, pub^*, b_1, x) = \mathsf{adjust}(2, key_2, pub^*, b_2, x)$, then $\mathsf{adjust}(1, key_1, pub^*, b_1, x') = \mathsf{adjust}(2, key_2, pub^*, b_2, x')$ as well, for every prefix x' of x. This follows trivially by inspection and the recursive nature of adjust. Note that the same b_1, b_2 are used for both x and x'.

Observation: Let $pub^* = (pub, \vec{C})$. Suppose $\mathsf{adjust}(1, key_1, pub^*, b_1, x) = \mathsf{adjust}(2, key_2, pub^*, b_2, x)$, and $\mathsf{IDPF.Eval}(1, key_1, pub, x) = (\_, y_1, R_1)$, and $\mathsf{IDPF.Eval}(2, key_2, pub, x) = (\_, y_2, R_2)$. Then:

1. If $b_1 = b_2$ then $\mathsf{RG}(x \,\|\, -y_1 \,\|\, -R_1) = \mathsf{RG}(x \,\|\, y_2 \,\|\, R_2)$. This includes the case where $(y_1, R_1) + (y_2, R_2) = (0, 0)$, making the two calls to RG identical. It also includes the case where these two calls to RG are a collision.

2. If $b_1 \neq b_2$ then $\vec{C}[|x|] = \mathsf{RG}(x \,\|\, -y_1 \,\|\, -R_1) \oplus \mathsf{RG}(x \,\|\, y_2 \,\|\, R_2)$.

This observation can be verified by inspection.

Let $\mathcal{E}_1$ denote the bad event that the adversary queries RG and observes a collision. If RG has outputs of length 4κ, and the adversary makes q oracle queries, then the probability of this bad event is bounded by $q^2/2^{4\kappa}$. When $\mathcal{E}_1$ does *not* happen, then in condition (1) above, only the case that $(y_1, R_1) + (y_2, R_2) = (0, 0)$ is possible.

Let $\mathcal{E}_2$ denote the bad event that the adversary makes any four queries to RG that satisfy:

$$\mathsf{RG}(pfx \,\|\, -d_1 \,\|\, -R_1) \oplus \mathsf{RG}(pfx \,\|\, d_2 \,\|\, R_2) = \\ \mathsf{RG}(pfx' \,\|\, -d_1' \,\|\, -R_1') \oplus \mathsf{RG}(pfx' \,\|\, d_2' \,\|\, R_2')$$

for $pfx \neq pfx'$ and $d_1 + d_2 \neq 0$ and $d_1' + d_2' \neq 0$. (These conditions ensure that the four calls to RG must be on distinct inputs.) If RG has outputs of length 4κ, and the adversary makes q oracle queries, then the probability of this bad event is bounded by $q^4/2^{4\kappa}$. When $\mathcal{E}_2$ does *not* happen, then any value $C \in \{0,1\}^{4\kappa}$ uniquely determines *at most one* pair of queries satisfying $C = \mathsf{RG}(pfx \,\|\, -d_1 \,\|\, -R_1) \oplus \mathsf{RG}(pfx \,\|\, d_2 \,\|\, R_2)$

We can apply the two observations inductively and obtain the following. If $\mathsf{adjust}(1, key_1, pub^*, b_1, x) = \mathsf{adjust}(2, key_2, pub^*, b_2, x)$ for $b_1 \neq b_2$, then every correction word $\vec{C}[\ell]$ must be of the form $\mathsf{RG}(x[1:\ell] \,\|\, \cdots) \oplus \mathsf{RG}(x[1:\ell] \,\|\, \cdots)$, for $\ell \leq |x|$. Then, provided that $\mathcal{E}_2$ does not happen, there is at most one x of length ℓ for which $\vec{C}$ can be written in this way.

Combining all of these observations, we can define the extractor as shown in Figure 10.

Conditioned on the event that E doesn't abort (which happens only with probability $(q^4 + q^2)/2^{4\kappa}$), we claim that the adversary has no advantage in the extractability game.

Consider a query to $\underline{\mathsf{Eval}}(\vec{x})$ in the game, and assume the call to Verify succeeds. Then for every $\vec{x}[i]$, the corresponding calls to adjust produce identical output. If these calls to adjust have $b_1 = b_2$, and $\mathcal{E}_1$ has not happened, then the corresponding output $\vec{y}[i]$ must be 0. If these calls to adjust have $b_1 \neq b_2$, then $\vec{C}[\ell]$ must have the form $\mathsf{RG}(\vec{x}[i] \,\|\, \cdots) \oplus \mathsf{RG}(\vec{x}[i] \,\|\, \cdots)$. If $\mathcal{E}_2$ has not happened, then $\vec{x}[i]$ is in fact unique with this property, and therefore $\vec{x}[i]$ is a prefix of α computed by the extractor E. One can easily check that E

extracts $\vec{\beta}[\ell]$ that is equal to the VIDPF output $\vec{y}[i]$. In other words, $\vec{y}$ matches the output of $f_{\alpha,\vec{\beta}}$. Hence, the adversary's advantage is zero.

Privacy: Let S^{IDPF} be the simulator for privacy for the underlying IDPF. The simulator for our construction is given in Figure 11. We prove privacy in a series of hybrids, also illustrated in Figure 11. Game $\underline{\mathsf{G}}_0$ refers to the original experiment $\mathsf{Exp}^{\mathrm{priv}}_{\mathsf{VIDPF}}$, where we have inlined the definition of S_2 for convenience. The $b = 0$ and $b = 1$ branches of the $\underline{\mathsf{Sketch}}$ oracle differ only in whose shares are given as input to $\mathsf{VIDPF.VEval}$. By the correctness of the scheme, the distinction doesn't matter, so the $\underline{\mathsf{Sketch}}$ oracle is independent of b. Eliminating the conditional in the $\underline{\mathsf{Sketch}}$ oracle, we obtain $\underline{\mathsf{G}}_1$, which is distributed identically to $\underline{\mathsf{G}}_0$.

$\underline{\mathsf{G}}_2$ is identical to $\underline{\mathsf{G}}_1$, but we have inlined the definition of $\mathsf{VIDPF.Gen}$ for convenience. By the correctness of the underlying IDPF, outputs of $\mathsf{IDPF.Eval}(1, \cdot)$ and $\mathsf{IDPF.Eval}(2, \cdot)$ are secret-shares of the appropriate plaintext values. So it has no effect on the adversary's view to solve for the output of $\mathsf{IDPF.Eval}(3 - \hat{\jmath}, \cdot)$ using the plaintext values and the output of $\mathsf{IDPF.Eval}(\hat{\jmath}, \cdot)$, instead of using $key_{3-\hat{\jmath}}$. In doing so, we obtain $\underline{\mathsf{G}}_3$ which is distributed identically to $\underline{\mathsf{G}}_2$.

Now notice that in $\underline{\mathsf{G}}_3$, the value $key_{3-\hat{\jmath}}$ is never used. As such, we can replace line 26 (the call to $\mathsf{IDPF.Gen}$) with a corresponding call to the simulator S^{IDPF}, which generates a simulated $key_{\hat{\jmath}}$ and pub. Call the result $\underline{\mathsf{G}}_4$ (not pictured); this advantage in distinguishing $\underline{\mathsf{G}}_3$ from $\underline{\mathsf{G}}_4$ is at most ϵ_{priv}.

In $\underline{\mathsf{G}}_4$, the random values $\vec{R}[\ell]$ are used only to solve for $R_{3-\hat{\jmath}}$, which is in turn used only as an argument to RG. Define a bad event that the adversary ever queries RG at an input of this form — i.e., of the form $\mathsf{RG}(\cdot \,\|\, \cdot \,\|\, \vec{R}[\ell] - R_{\hat{\jmath}})$. The probability of the bad event is bounded by $q/2^{\kappa}$ since $\vec{R}[\ell]$ is uniformly random. Conditioned on this bad event not happening, the results of these queries to RG are freshly random, and the value that is assigned to $\vec{C}[\ell]$ is uniform. In that case, the behavior of the game is independent of the challenge bit because S_1 also assigns uniform values to $\vec{C}[\ell]$. The advantage in guessing the challenge bit is therefore bounded by the probability of the bad event. □

B Instantiating Delayed-Input FLP

Our main result is to construct a delayed-2-input FLP for use in Doplar.

Lemma 2. *The construction in Figure 12 (when suitably instantiated) is a delayed-2-input FLP with perfect completeness, soundness* $4(n + 2)/(|\mathbb{F}| - n - 2)$*, and privacy* $1/|\mathbb{F}|$*, for* $\mathbb{F}$*-arithmetic circuits with* n *multiplication gates.*

The main idea of the construction is simple. The prover wishes to generate a proof that will work with either of two instances x_1 and x_2. She simply generates a separate FLP proof for both instances x_1 and x_2, and randomly permutes the two proofs. To verify the combined proof against some x, the verifier accepts iff either of the component proofs verifies against that x.

Completeness and soundness of this construction are relatively clear. However, the construction is not necessarily zero-knowledge. While verifying the combined proof, we expect to verify a component proof against a *proof that was generated for some other instance* — e.g., verify a proof generated for x_1 against x_2. The standard zero-knowledge property of the underlying FLP does not apply to this situation. Indeed, since the Query function is linear, the result of querying a "mismatched" instance-proof pair will reveal "how far away" the instance is from the correct one.

We show that, when the underlying FLP is that of Boneh et al. [17], and extra randomness is introduced into the statement by means of the $\mathsf{Encode}(\Delta, \cdot)$ function, even the queries to the "mismatched" instance+proof can be simulated. Intuitively, the extra uncertainty of Δ blinds the results of the problematic queries.

Proof of Lemma 2. Figure 12 describes a delayed-input FLP that uses a basic FLP as a building block. Our claims in this proof rely on that FLP being instantiated using the construction of [17], also used in the VDAF draft specification (see [9, Section 7.3]). We recall the relevant aspects of that construction below, as needed.

```
$S_1(\hat{j})$:
  $(key, pub) \leftarrow S^{\mathsf{IDPF}}()$
  for $\ell \in [\eta]$: $\vec{C}[\ell] \leftarrow\!\$\ \{0,1\}^{4\kappa}$
  $pub^* \leftarrow (pub, \vec{C})$
  ret $(key, pub^*)$

$S_2(\hat{j}, key, pub, \vec{x})$:
  $(h, \_) \leftarrow \mathsf{VIDPF.VEval}(\hat{j}, key, pub, \vec{x})$
  ret $h$
```

```
Game $\mathsf{G}_0$ $\boxed{\mathsf{G}_1}$:
  $b \leftarrow\!\$\ \{0,1\}$
  $(st_A, \alpha, \vec{\beta}, \hat{j}) \leftarrow A(\,)$
  if $b = 0$: $(key_{\hat{j}}, pub^*) \leftarrow\!\$\ S_1(\hat{j})$
  else: $(key_1, key_2, pub^*) \leftarrow\!\$\ \mathsf{VIDPF.Gen}(\alpha, \vec{\beta})$
  $b^* \leftarrow A^{\mathsf{Sketch}}(st_A, key_{\hat{j}}, pub^*)$
  ret $b = b^*$

$\mathsf{Sketch}(\vec{x})$:
  if $b = 0$:
    // $h \leftarrow S_2(\hat{j}, key_{\hat{j}}, pub^*, \vec{x})$
    $(h, \_) \leftarrow \mathsf{VIDPF.VEval}(\hat{j}, key_{\hat{j}}, pub^*, \vec{x})$
  else: $(h, \_) \leftarrow \mathsf{VIDPF.VEval}(3 - \hat{j}, key_{3-\hat{j}}, pub^*, \vec{x})$
  ret $h$
```

```
Game $\mathsf{G}_2$ $\boxed{\mathsf{G}_3}$:
  $b \leftarrow\!\$\ \{0,1\}$
  $(st_A, \alpha, \vec{\beta}, \hat{j}) \leftarrow A(\,)$
  if $b = 0$: $(key_{\hat{j}}, pub^*) \leftarrow\!\$\ S_1(\hat{j})$
  else:
    // $(key_1, key_2, pub^*) \leftarrow\!\$\ \mathsf{VIDPF.Gen}(\alpha, \vec{\beta})$:
    for $\ell \in [\eta]$:
      $\vec{R}[\ell] \leftarrow\!\$\ \{0,1\}^{\kappa}$
      $\vec{\beta}^*[\ell] \leftarrow (1, \vec{\beta}[\ell], \vec{R}[\ell])$
    $(key_1, key_2, pub) \leftarrow \mathsf{IDPF.Gen}(\alpha, \vec{\beta}^*)$
    for $\ell \in [\eta]$:
      $pfx \leftarrow \alpha[1:\ell]$
      $(b_1, data_1, R_1) \leftarrow \mathsf{IDPF.Eval}(1, key_1, pub, pfx)$
      $(b_2, data_2, R_2) \leftarrow \mathsf{IDPF.Eval}(2, key_2, pub, pfx)$
      $\boxed{(b_{\hat{j}}, data_{\hat{j}}, R_{\hat{j}}) \leftarrow \mathsf{IDPF.Eval}(\hat{j}, key_{\hat{j}}, pub, pfx)}$
      $\boxed{(b_{3-\hat{j}}, data_{3-\hat{j}}, R_{3-\hat{j}}) \leftarrow (1 \oplus b_{\hat{j}}, \vec{\beta}[\ell] - data_{\hat{j}}, \vec{R}[\ell] - R_{\hat{j}})}$
      $\vec{C}[\ell] \leftarrow \mathsf{RG}(pfx \parallel -data_1 \parallel R_1)$
            $\oplus\, \mathsf{RG}(pfx \parallel data_2 \parallel R_2)$
    $pub^* \leftarrow (pub, \vec{C})$
  $b^* \leftarrow A^{\mathsf{Sketch}}(st_A, key_{\hat{j}}, pub^*)$
  ret $b = b^*$

$\mathsf{Sketch}(\vec{x})$:
  // $h \leftarrow S_2(\hat{j}, key_{\hat{j}}, pub^*, \vec{x})$
  $(h, \_) \leftarrow \mathsf{VIDPF.VEval}(\hat{j}, key_{\hat{j}}, pub^*, \vec{x})$
  ret $h$
```

Figure 11: Simulator and hybrids used in the proof of privacy for the VIDPF construction.

In Doplar, we will use our DFLP construction for the language $\mathcal{L} = \{0,1\}$ — i.e., we use it to prove that a value is zero or one. In this case, we instantiate the underlying FLP with the circuit:

$$C((s,t,\mathit{\Delta}), r) = \big(r \cdot s(s-1) + r^2 \cdot (s \cdot \mathit{\Delta} - t)\big)^2\,, \tag{1}$$

where r denotes the joint randomness. This circuit recognizes the set of inputs $(s,t,\mathit{\Delta}) \in \{(0,0,\mathit{\Delta}), (1,\mathit{\Delta},\mathit{\Delta})\}$. Note that FLP has input length $n = 3$ and joint-randomness length $jl = 1$; its circuit has 3 multiplication gates ($s(s-1)$, $s\mathit{\Delta}$, and the outer square). In the more general case, FLP will be instantiated for the language $\{(s,t,\mathit{\Delta}) \mid s \in \mathcal{L} \wedge s\mathit{\Delta} = t\}$. If the circuit for membership in $\mathcal{L}$ has n multiplication gates, then FLP will be instantiated with a circuit with $n+2$ multiplication gates.

Completeness follows immediately from the perfect completeness of the underlying FLP. The FLP of [17] has soundness $2n'/(|\mathbb{F}| - n')$ when its circuit has n' multiplication gates. We instantiate that FLP with $n' = n+2$, and we also incur a factor 2 loss in soundness since our construction verifies two proofs in the underlying FLP. Hence, we obtain the soundness bound stated in the lemma.

The zero-knowledge simulator for our construction is given as S in Figure 13. To demonstrate privacy, we first consider the hybrid on the left of Figure 13. With the gray box included and white box excluded, the hybrid generates exactly the honest verifier's view. In this game, both proofs are queried on x_c, the adversary's choice. Note that proof $\pi_{b\oplus c}$ was generated with input x_c in mind, while $\pi_{b\oplus c\oplus 1}$ was not. Let $u = b \oplus c \oplus 1$, the index of the "mismatched" proof (i.e., π_u was generated with $x_{c\oplus 1}$ in mind, not x_c). By

```
$\mathsf{DFLP}^*.\mathsf{Prove}(\{\vec{x}_1, \vec{x}_2\}, \mathit{\Delta}, jr)$:
  $(jr_1, jr_2) \leftarrow jr$
  $\vec{e}_1 \leftarrow \mathsf{FLP.Encode}(\mathit{\Delta}, \vec{x}_1)$
  $\vec{e}_2 \leftarrow \mathsf{FLP.Encode}(\mathit{\Delta}, \vec{x}_2)$
  $b \leftarrow\!\!\$\ \{1, 2\}$
  $\pi_b \leftarrow\!\!\$\ \mathsf{FLP.Prove}(\vec{e}_1, \mathit{\Delta}, jr_b)$
  $\pi_{3-b} \leftarrow\!\!\$\ \mathsf{FLP.Prove}(\vec{e}_2, \mathit{\Delta}, jr_{3-b})$
  ret $(\pi_1, \pi_2)$

$\mathsf{DFLP}^*.\mathsf{Query}(\vec{e}, \mathit{\Delta}, (\pi_1, \pi_2), jr; qr)$:
  $(jr_1, jr_2) \leftarrow jr$; $(qr_1, qr_2) \leftarrow qr$
  $\sigma_1 \leftarrow\!\!\$\ \mathsf{FLP.Query}(\vec{e}, \mathit{\Delta}, \pi_1, jr_1; qr_1)$
  $\sigma_2 \leftarrow\!\!\$\ \mathsf{FLP.Query}(\vec{e}, \mathit{\Delta}, \pi_2, jr_2; qr_2)$
  ret $(\sigma_1, \sigma_2)$

$\mathsf{DFLP}^*.\mathsf{Decide}(\sigma)$:
  $(\sigma_1, \sigma_2) \leftarrow \sigma$
  ret $\mathsf{FLP.Decide}(\sigma_1)$
      $\vee\ \mathsf{FLP.Decide}(\sigma_2)$

$\mathsf{DFLP}^*.\mathsf{Encode}(\mathit{\Delta} \in \mathbb{F}, \vec{x} \in \mathbb{F}^n)$:
  for $i \in [n]$:
    $\vec{e}[i] \leftarrow \vec{x}[i]$
    $\vec{e}[i+n] \leftarrow \mathit{\Delta} \cdot \vec{x}[i]$
  ret $\vec{e}$

$\mathsf{DFLP}^*.\mathsf{Decode}(\vec{e} \in \mathbb{F}^{2n})$:
  ret $\vec{e}[1:n]$
```

Figure 12: Delayed-2-input FLP construction $\mathsf{DFLP}^*[\mathsf{FLP}]$. The construction should be instantiated where FLP is the FLP for arithmetic circuits from [17].

applying the linearity of $\mathsf{Encode}(\mathit{\Delta}, \cdot)$ and $\mathsf{Query}(\cdot, \cdot, \cdot)$, we can write:

$$\begin{aligned}\mathsf{Query}(\mathsf{Encode}(\mathit{\Delta}, x_c), &\mathit{\Delta}, \pi_u, jr_u; qr_u) = \\ &\mathsf{Query}(\mathsf{Encode}(\mathit{\Delta}, x_{c\oplus 1}), \mathit{\Delta}, \pi_u, jr_u; qr_u) + \mathsf{Query}(\mathsf{Encode}(\mathit{\Delta}, x_c - x_{c\oplus 1}), 0, \vec{0}, jr_u; qr_u)\end{aligned}$$

Making this change of notation in the game yields hybrid $\underline{\mathsf{G}}_1$ (in Figure 13, gray boxes excluded and outlined box included). $\underline{\mathsf{G}}_1$ is distributed identically to the original privacy game.

In each call to Query in $\underline{\mathsf{G}}_1$ that involves a value π_i, we use the same input that was used to generate π_i. Hence, we can apply the zero-knowledge property of the underlying FLP to each such expression. In doing so, we obtain the hybrid $\underline{\mathsf{G}}_2$ on the right of Figure 13. The underlying FLP of [17] has perfect zero-knowledge, so $\underline{\mathsf{G}}_2$ is distributed identically to the original game.

To complete the proof, it suffices to show that σ_u is distributed pseudorandomly in $\mathbb{F}^3 \times \{0\}$, since the simulator samples σ_u uniformly from that set. In particular, when $\widetilde{\sigma}$ is distributed as in a simulated proof, $\mathit{\Delta}$ is random, and $d \neq 0$, what is the distribution on $\widetilde{\sigma} + \mathsf{Query}(\mathsf{Encode}(\mathit{\Delta}, d), \vec{0}, \vec{0}, \cdots)$?

To answer this question, we must use specific properties of the FLP from [17]. We first briefly review the main idea behind their proof. The prover defines two polynomials L and R such that, for each multiplication gate i in the verification circuit, the value on its left wire is $L(i)$ and its right wire $R(i)$. Additionally, $L(0)$ and $R(0)$ are chosen uniformly. Define the "gadget" polynomial $G = L \times R$ — then $G(i)$ is the value of the output wire of the ith gate.

The proof vector π then consists of $L(0)$, $R(0)$, and the coefficients of the G polynomial. With that in mind, the Query algorithm makes 4 linear queries to the input + proof vector:

1. Obtain evaluations of the polynomial L as follows:

   - $L(0)$ is part of the proof vector.
   - For $i > 0$, if the left input to gate i is an input to the circuit, then $L(i)$ is given as part of the proof input/instance, to which Query has access.
   - Otherwise, the left input to gate i is the output of some other multiplication gate j. This value can be obtained as $G(j)$, since the coefficients of G are included in the proof vector.

   Reconstruct L as the result of Lagrange interpolation over the points $\{(i, L(i))\}$. Evaluate this polynomial L at point qr (the query randomness).

2. Similarly, reconstruct R and evaluate it at point qr.

3. Evaluate the polynomial G at point qr.

4. Evaluate G at point m, where the output of verification circuit is the output wire of the m'th multiplication gate.

Suppose the results of these queries are (r, s, t, u); the Decide algorithm checks that $t = rs$ and $u = 0$. The zero-knowledge property is that the result of the queries is distributed as $(r, s, rs, 0)$ for uniform $r, s \leftarrow\!\!\$\ \mathbb{F}$.

With Query as above, we now consider the distribution of

$$(r, s, rs, 0) + \mathsf{Query}(\mathsf{Encode}(\varDelta, d), \vec{0}, \vec{0}, \cdots),$$

where $\varDelta, r, s$ are uniform in $\mathbb{F}$.

- The first component of this expression is uniform due to r.
- With overwhelming probability $1 - 1/|\mathbb{F}|$ we have $r \neq 0$. Conditioned on $r \neq 0$, the third component of the expression is uniform, since it is masked with rs, and s is uniform (even conditioned on the first component).
- Let q_4 be the 4th component of Query's output in the above expression. By definition of Query, q_4 is the result of evaluating G at point qr. But in this expression, the "proof vector" argument to Query is all zeroes, hence Query evaluates the all-zeroes polynomial and outputs $q_4 = 0$. Hence the 4th component of the overall expression is zero.
- Let q_2 be the second output of Query in the above expression. We see that q_2 is the result of evaluating polynomial R at point qr, after reconstructing R as described above. Fix a position i in which $\vec{d}[i] \neq 0$. Then the $(n + i)$th position of $\vec{e} = \mathsf{Encode}(\varDelta, \vec{d})$ is $\vec{e}[i] = \vec{d}[i]\varDelta$, and therefore is uniformly distributed when $\varDelta$ is uniformly distributed.

  The final multiplication gate in the verification circuit is the outermost square in (1). The input to this squaring operation is a linear combination that includes $\vec{e}[i]$. So as $\vec{e}[i]$ is uniformly distributed, the input to this multiplication gate is also uniformly distributed. Then the result of interpolating polynomial R (based on $\vec{e}[i]$ among other values) and evaluating R at qr is also uniformly distributed. In other words, q_2 is uniformly distributed over uniform choice of $\varDelta$, so the second component of the above expression is uniform.

Overall, we have shown that the distribution of σ_u in $\underline{\mathsf{G}}_2$ of Figure 13 is statistical distance $1/|\mathbb{F}|$ from the simulator's distribution: uniform over $\mathbb{F}^3 \times \{0\}$. Hence in $\underline{\mathsf{G}}_2$ the adversary has advantage bounded by $1/|\mathbb{F}|$ in $\underline{\mathsf{G}}_2$. □

C Proofs of Theorems

C.1 Prio3 Robustness (Theorem 1)

We begin by instantiating the robustness game for $\varPi$ in Figure 14. We expand the Prep algorithm and make a few simplifications to the game's internal notation and bookkeeping. First, the game $\mathsf{Exp}^{\text{robust}}$ calls for a VDAF with an arbitrary number of rounds, but Prio3 constructions has just one round. Second, we know that the Prep algorithm will be called exactly twice for each aggregator, and that the initial broadcast message and state are empty. We Therefore unroll the loop of lines 5–14 of Figure 3 and evaluate those if-statements whose values are pre-determined. Third, we replace table St with a vector $\vec{st}$ and remove table Msg altogether. (The transcript output by the oracle is now constructed at the end on line 21 on the left-hand panel of Figure 14.) Fourth, we evaluate Prep in parallel for all aggregators instead of in sequence; the order of these operations does not affect their results because aggregators do not share state. Fifth, we perform the deterministic Decide operation only once since its result is the same for all aggregators. Finally, we replace each call to RG_i with a call to the corresponding random oracle $\underline{\mathsf{RO}}_i$. Let q_i denote the number of queries A makes to $\underline{\mathsf{RO}}_i$; note that $q_{\mathsf{RG}} = q_1 + \cdots + q_7$.

We have also dropped winning condition on line 16 of Figure 3. By definition, $\varPi.\mathsf{refineFromShares}(\varepsilon, \vec{x}) = \varPi.\mathsf{Unshard}(1, (\varPi.\mathsf{Agg}(\vec{inp}[1]), \ldots, \varPi.\mathsf{Agg}(\vec{inp}[s])))$, where $inp[\hat{j}]$ is the unpacked inner measurement share of

```
$\underline{\mathsf{Exp}^{\mathrm{priv}}_{\mathsf{DFLP},S}(A)}$:  $\boxed{\underline{\mathsf{G}}_1(A)}$:
$b \leftarrow_\$ \{0,1\}$
$(\{x_0, x_1\}, st_A) \leftarrow A()$
if $b = 0$:
  $(st_S, (jr_0, jr_1), (qr_0, qr_1)) \leftarrow S_1()$
else:
  $jr_0, jr_1 \leftarrow_\$ \mathbb{F}^{jl}$; $qr_0, qr_1 \leftarrow_\$ \mathbb{F}^{ql}$
  $\Delta \leftarrow_\$ \mathbb{F}$
  // $\mathsf{Prove}(\{x_0, x_1\}, \Delta, jr)$
  $b \leftarrow_\$ \{0,1\}$
  $\pi_b \leftarrow \mathsf{Prove}(\mathsf{Encode}(\Delta, \vec{x}_0), \Delta, jr_b)$
  $\pi_{1\oplus b} \leftarrow \mathsf{Prove}(\mathsf{Encode}(\Delta, \vec{x}_1), \Delta, jr_{1\oplus b})$
$(c, st_A) \leftarrow A(st_A, (jr_0, jr_1), (qr_0, qr_1))$
if $b = 0$: $(\sigma_0, \sigma_1) \leftarrow S^*(st_S)$
else: // $\mathsf{Query}(\mathsf{Encode}(\Delta, x_c), \Delta, \pi, jr;\ qr)$
  $\sigma_0 \leftarrow \mathsf{Query}(\mathsf{Encode}(\Delta, x_c), \Delta, \pi_0, jr_0; qr_0)$
  $\sigma_1 \leftarrow \mathsf{Query}(\mathsf{Encode}(\Delta, x_c), \Delta, \pi_1, jr_1; qr_1)$
  $\sigma_b \leftarrow \mathsf{Query}(\mathsf{Encode}(\Delta, x_0), \Delta, \pi_b, jr_b; qr_b)$
  $\sigma_{1\oplus b} \leftarrow \mathsf{Query}(\mathsf{Encode}(\Delta, x_1), \Delta, \pi_{1\oplus b}, jr_{1\oplus b}; qr_{1\oplus b})$
  $u \leftarrow b \oplus c \oplus 1$ // index of "mismatched" proof
  $\sigma_u \leftarrow \sigma_u$
      $+ \mathsf{Query}(\mathsf{Encode}(\Delta, x_c - x_{1\oplus c}), 0, \vec{0}, jr_u; qr_u)$
$b' \leftarrow A(st_A, (\sigma_0, \sigma_1))$
ret $b == b'$
```

```
$\underline{\mathsf{G}_2(A)}$:
$b \leftarrow_\$ \{0,1\}$
$(\{x_0, x_1\}, st_A) \leftarrow A()$
if $b = 0$:
  $(st_S, (jr_0, jr_1), (qr_0, qr_1)) \leftarrow S_1()$
else:
  $\Delta \leftarrow_\$ \mathbb{F}$
  $b \leftarrow_\$ \{0,1\}$
  $(jr_0, qr_0, \sigma_0) \leftarrow S_{\mathsf{FLP}}()$
  $(jr_1, qr_1, \sigma_1) \leftarrow S_{\mathsf{FLP}}()$
$(c, st) \leftarrow A(st_A, (jr_0, jr_1), (qr_0, qr_1))$
if $b = 0$: $(\sigma_0, \sigma_1) \leftarrow S^*(st_S)$
else:
  $u \leftarrow b \oplus c \oplus 1$ // index of "mismatched" proof
  $\sigma_u \leftarrow \sigma_u$
      $+ \mathsf{Query}(\mathsf{Encode}(\Delta, x_c - x_{1\oplus c}), 0, \vec{0}, jr_u; qr_u)$
$b' \leftarrow A(st_A, (\sigma_0, \sigma_1))$
ret $b == b'$

$\underline{S_1()}$:
$(jr_0, qr_0, \sigma_0) \leftarrow S()$
$(jr_1, qr_1, \sigma_1) \leftarrow S()$
ret $\big(st_S = (\sigma_0, \sigma_1), (jr_0, jr_1), (qr_0, qr_1)\big)$
$\underline{S_2(\sigma_0, \sigma_1)}$:
$b \leftarrow_\$ \{0,1\}$
$\sigma_b \leftarrow_\$ \mathbb{F}^3 \times \{0\}$ // overwrite $\sigma_b$
ret $(\sigma_0, \sigma_1)$
```

Figure 13: Hybrids for zero-knowledge property of the delayed-2-input FLP construction.

input share $\vec{x}[\hat{j}]$ for each $\hat{j}$. Thus w can never be set by forcing the refined shares to mismatch the expected refined measurement.

Now we express the proof with a series of incrementally changed games, beginning with $\underline{\mathsf{G}}_1$ (c.f. Figure 14). The joint randomness for each aggregator $\hat{j}$ is derived in $\mathsf{Exp}^{\mathrm{robust}}_{\Pi}$ from the seed $jseed_{\hat{j}}$ of that aggregator, which is also the state $\vec{st}[\hat{j}]$. In $\underline{\mathsf{G}}_1$, we instead derive joint randomness from $\vec{st}[1]$ for all aggregators, thus ensuring that the joint randomness is the same for everyone.

We build a wrapper adversary B for which

$$\mathsf{Adv}^{\mathrm{robust}}_{\Pi}(A) \leq \Pr[\underline{\mathsf{G}}_1(B)] + \frac{q_5}{2^\kappa}\,. \tag{2}$$

Adversary B only makes queries to $\underline{\mathsf{Prep}}$ that set $\vec{st}[\hat{j}] = \vec{st}[1]$ for all $\hat{j}$. It accomplishes this by calculating $\vec{st}[\hat{j}]$ for every aggregator and $\underline{\mathsf{Prep}}$ query made by A. If it finds that $\vec{st}[\hat{j}] = \vec{st}[1]$ for all aggregators, it forwards the query to its own $\underline{\mathsf{Prep}}$ oracle. Otherwise, it runs $\underline{\mathsf{Prep}}$ itself. B can perfectly simulate $\underline{\mathsf{Prep}}$ except for line 9, because it does not know sk. Instead, B picks its own verification key sk' and uses sk' in line 9 where $\underline{\mathsf{Prep}}$ would use sk. Adversary A can detect the substitution of sk' for sk in two cases: If A queries $\underline{\mathsf{RO}}_5$ on seed sk'; or if the $\underline{\mathsf{Prep}}$ oracle and B query $\underline{\mathsf{RO}}_5$ on the same context string. The latter event does not occur because each query to $\underline{\mathsf{RO}}_5$ contains a unique nonce. The former occurs with probability at most $\frac{q_5}{2^\kappa}$, because sk' is a uniformly random κ-bit string. The queries that B simulates would always set $acc_{\hat{j}} \leftarrow 0$ in line 17. Thus any query that would set $w \leftarrow \texttt{true}$ is forwarded to the $\underline{\mathsf{Prep}}$ oracle by B, and B wins whenever A does. The claim follows.

Next, we use the full linearity of FLP to decompose $\mathsf{FLP.Query}$ into algorithm Q and a matrix multiplication operation, as shown in the left-hand panel of Figure 15. Q is a randomized algorithm, but it is executed deterministically with fixed input jr and coins qr. We may therefore call Q only once to eliminate redun-

```
Game $\mathsf{Exp}^{\mathrm{robust}}_{\Pi}(A)$ $\boxed{\underline{\mathsf{G}}_1(A)}$ :
 1 $w \leftarrow \texttt{false};\ sk \leftarrow\!\!\$\ \{0,1\}^{\kappa}$
 2 $A^{\underline{\mathsf{RO}},\underline{\mathsf{Prep}}}(\,)$; ret $w$

$\underline{\mathsf{Prep}}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$:
 3   if Used$[n] \neq \bot$: ret $\bot$
 4   Used$[n] \leftarrow \top$
 5   for $\hat{j} \in [s]$:
 6     $(\vec{inp}[\hat{j}], \vec{\pi}[\hat{j}], blind) \leftarrow \mathsf{Unpack}(\hat{j}, \vec{x}[\hat{j}])$
 7     $(\vec{\rho}, ) \leftarrow \vec{msg};\ \vec{\rho}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_7(blind, \hat{j} \,\|\, n \,\|\, \vec{inp}[\hat{j}])$
 8     $\vec{rseed}[\hat{j}] \leftarrow \vec{\rho}[\hat{j}]$
 9     $\vec{st}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_6(0^{\kappa}, \vec{\rho})$ //joint rand seed
10     $jr \leftarrow \underline{\mathsf{RO}}_1(\vec{st}[\hat{j}], \varepsilon)$ $\boxed{jr \leftarrow \underline{\mathsf{RO}}_1(\vec{st}[1], \varepsilon)}$
11     $qr \leftarrow \underline{\mathsf{RO}}_5(sk, n)$
12     $\vec{vfs}[\hat{j}] \leftarrow \mathsf{Query}(\vec{inp}[\hat{j}], \vec{\pi}[\hat{j}], jr; qr)$
13   $vf \leftarrow \sum_{\hat{j}=1}^{s} \vec{vfs}[\hat{j}]$
14   $d \leftarrow \mathsf{FLP.Decide}(vf)$
15   for $\hat{j} \in [s]$:
16     $jseed_{\hat{j}} \leftarrow \vec{st}[\hat{j}];\ jseed'_{\hat{j}} \leftarrow \underline{\mathsf{RO}}_6(0^{\kappa}, \vec{rseed})$
17     $acc_{\hat{j}} \leftarrow d \wedge [[jseed_{\hat{j}} = jseed'_{\hat{j}}]]$
18     $w \leftarrow (w \vee [acc_{\hat{j}} \wedge \mathsf{refineFromShares}(\varepsilon, \vec{x}) \notin \mathcal{L}])$
19   ret $(w, (msg_{\mathrm{Init}}, (\vec{vfs}[\hat{j}], \vec{rseed}[\hat{j}]))_{\hat{j} \in [s]})$
```

```
Adversary $B^{\underline{\mathsf{RO}},\underline{\mathsf{Prep}}}(\,)$:
 1 $sk' \leftarrow\!\!\$\ \{0,1\}^{\kappa}$
 2 $A^{\underline{\mathsf{RO}},\mathsf{PrepSim}}(\,)$

$\mathsf{PrepSim}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$:
 3   if Used$[n] \neq \bot$: ret $\bot$
 4   Used$[n] \leftarrow \top;\ fwd \leftarrow \texttt{true}$
 5   for $\hat{j} \in [s]$:
 6     $(\vec{inp}[\hat{j}], \vec{\pi}[\hat{j}], blind) \leftarrow \mathsf{Unpack}(\hat{j}, \vec{x}[\hat{j}])$
 7     $(\vec{\rho}, ) \leftarrow \vec{msg};\ \vec{\rho}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_7(blind, \hat{j} \,\|\, n \,\|\, \vec{inp}[\hat{j}])$
 8     $\vec{rseed}[\hat{j}] \leftarrow \vec{\rho}[\hat{j}]$
 9     $\vec{st}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_6(0^{\kappa}, \vec{\rho})$ // Joint rand seed
10     if $\vec{st}[\hat{j}] \neq \vec{st}[1]$: $fwd \leftarrow \texttt{false}$
11     $jr \leftarrow \underline{\mathsf{RO}}_1(\vec{st}[\hat{j}], \varepsilon)$
12     $qr \leftarrow \underline{\mathsf{RO}}_5(sk', n)$
13     $\vec{vfs}[\hat{j}] \leftarrow \mathsf{Query}(\vec{inp}[\hat{j}], \vec{\pi}[\hat{j}], jr; qr)$
14   if $fwd$: return $\underline{\mathsf{Prep}}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$
15   ret $(\texttt{false}, (msg_{\mathrm{Init}}, (\vec{vfs}[\hat{j}], \vec{rseed}[\hat{j}]))_{\hat{j} \in [s]})$
```

Figure 14: Left: Definition of game $\underline{\mathsf{G}}_1$ for the proof of Theorem 1. Also shown is the robustness game for Π and adversary A with some simplifications applied. Right: Adversary B.

dancy. Finally, we sum the vectors $\vec{x}_{\hat{j}} \,\|\, \pi_{\hat{j}}$ before the multiplication instead of multiplying then summing the products. This preserves the output thanks to the associativity of matrix multiplication.

Full linearity is an information theoretic property that holds unconditionally for all inputs, proofs, and coins, so the new game computes the same verifier string vf. Thus

$$\Pr[\underline{\mathsf{G}}_1(B)] = \Pr[\underline{\mathsf{G}}_2(B)]\,. \tag{3}$$

We produce the next modified game, in the right-hand panel of Figure 15, to define variables $inp = \sum_{\hat{j}=1}^{s} \vec{inp}[\hat{j}]$ and $\pi = \sum_{\hat{j}=1}^{s} \vec{\pi}[\hat{j}]$ and invoke $\mathsf{PRG.Query}$ on inp, π directly. In addition, from Figure 5, we can see that $inp = \Pi.\mathsf{refineFromShares}(\varepsilon, \vec{x})$, so we substitute inp into line 21. By the full linearity of FLP, we have that $Q(jr; qr) \cdot (inp \,\|\, \pi) = \mathsf{FLP.Query}(inp, \pi, jr; qr)$. Again, these operations do not affect the adversary's view of $\underline{\mathsf{Prep}}$, and

$$\Pr[\underline{\mathsf{G}}_2(B)] = \Pr[\underline{\mathsf{G}}_3(B)]\,. \tag{4}$$

In the next game, we replace the pseudorandom query randomness qr with a fresh random string that is implicitly sampled by Query. We bound the difference in advantage between games $\underline{\mathsf{G}}_3$ and $\underline{\mathsf{G}}_4$ via a reduction B' to the pseudorandomness of RG_5. The reduction honestly simulates $\underline{\mathsf{G}}_3$ except in line 11, where it queries its challenge oracle on n and sets qr to the response. Because every nonce is unique, these queries are all distinct. When the challenge oracle is a random function, this is a perfect simulation of $\underline{\mathsf{G}}_4$; otherwise it is a perfect simulation of $\underline{\mathsf{G}}_3$. Adversary B' makes q_{Prep} queries to its challenge oracle; when RG_5 is modeled as a random oracle, there is a maximum of q_5 random oracle queries.

The generic PRF advantage for a (q_5, q_{Prep})-query attacker against a random oracle with domain $\{0,1\}^{\kappa}$ is bounded by the probability $\frac{q_5}{2^{\kappa}}$ that the attacker makes a random oracle query containing sk. Thus

$$\Pr[\underline{\mathsf{G}}_3(B)] \leq \Pr[\underline{\mathsf{G}}_4(B)] + \frac{q_5}{2^{\kappa}}. \tag{5}$$

Our next game ($\underline{\mathsf{G}}_5$ defined in the right-hand panel of Figure 16) differs from $\underline{\mathsf{G}}_4$ as follows. We set a $\texttt{bad}$ flag and force the adversary to lose if it makes two queries to $\underline{\mathsf{Prep}}$ which derive their joint randomness from

```
Game $\underline{\mathsf{G}}_1(B)$ $\boxed{\underline{\mathsf{G}}_2(B)}$ :
$w \leftarrow \texttt{false};\ sk \leftarrow\!\!\$\ \{0,1\}^\kappa$
$B^{\underline{\mathsf{RO}},\underline{\mathsf{Prep}}}(\,)$; ret $w$

$\underline{\mathsf{Prep}}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$:
  if Used$[n] \neq \bot$: ret $\bot$
  Used$[n] \leftarrow \top$
  for $\hat{j} \in [s]$:
    $(\vec{inp}[\hat{j}], \vec{\pi}[\hat{j}], blind) \leftarrow \mathsf{Unpack}(\hat{j}, \vec{x}[\hat{j}])$
    $(\vec{\rho}, ) \leftarrow \vec{msg};\ \vec{\rho}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_7(blind, \hat{j} \,\|\, n \,\|\, \vec{inp}[\hat{j}])$
    $\vec{rseed}[\hat{j}] \leftarrow \vec{\rho}[\hat{j}]$
    $\vec{st}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \vec{\rho})$ //joint rand seed
    $jr \leftarrow \underline{\mathsf{RO}}_1(\vec{st}[1], \varepsilon);\ qr \leftarrow \underline{\mathsf{RO}}_5(sk, n)$
    $\vec{vfs}[\hat{j}] \leftarrow \mathsf{Query}(\vec{inp}[\hat{j}], \vec{\pi}[\hat{j}], jr; qr)$
  $vf \leftarrow \sum_{\hat{j}=1}^{s} \vec{vfs}[\hat{j}]$
  $\boxed{jr \leftarrow \underline{\mathsf{RO}}_1(\vec{st}[1], \varepsilon);\ qr \leftarrow \underline{\mathsf{RO}}_5(sk, n)}$
  $\boxed{Z \leftarrow Q(jr; qr)}$
  $\boxed{vf \leftarrow Z \cdot \sum_{\hat{j}=1}^{s} \vec{inp}[\hat{j}] \,\|\, \vec{\pi}[\hat{j}]}$
  $d \leftarrow \mathsf{FLP.Decide}(vf)$
  for $\hat{j} \in [s]$:
    $jseed_{\hat{j}} \leftarrow \vec{st}[\hat{j}];\ jseed'_{\hat{j}} \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \vec{rseed})$
    $acc_{\hat{j}} \leftarrow d \wedge [[jseed_{\hat{j}} = jseed'_{\hat{j}}]]$
    $w \leftarrow (w \vee [acc_{\hat{j}} \wedge \mathsf{refineFromShares}(\varepsilon, \vec{x}) \notin \mathcal{L}])$
  ret $(w, (msg_{\mathrm{Init}}, (\vec{vfs}[\hat{j}], \vec{rseed}[\hat{j}]))_{\hat{j}\in[s]})$
```

```
Game $\underline{\mathsf{G}}_2(B)$ $\boxed{\underline{\mathsf{G}}_3(B)}$ :
$w \leftarrow \texttt{false};\ sk \leftarrow\!\!\$\ \{0,1\}^\kappa$
$B^{\underline{\mathsf{RO}},\underline{\mathsf{Prep}}}(\,)$; ret $w$

$\underline{\mathsf{Prep}}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$:
  if Used$[n] \neq \bot$: ret $\bot$
  Used$[n] \leftarrow \top$
  for $\hat{j} \in [s]$:
    $(\vec{inp}[\hat{j}], \vec{\pi}[\hat{j}], blind) \leftarrow \mathsf{Unpack}(\hat{j}, \vec{x}[\hat{j}])$
    $(\vec{\rho}, ) \leftarrow \vec{msg};\ \vec{\rho}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_7(blind, \hat{j} \,\|\, n \,\|\, \vec{inp}[\hat{j}])$
    $\vec{rseed}[\hat{j}] \leftarrow \vec{\rho}[\hat{j}]$
    $\vec{st}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \vec{\rho})$ //joint rand seed
  $jr \leftarrow \underline{\mathsf{RO}}_1(\vec{st}[1], \varepsilon)$
  $qr \leftarrow \underline{\mathsf{RO}}_5(sk, n)$
  $Z \leftarrow Q(jr; qr)$
  $vf \leftarrow Z \cdot \sum_{\hat{j}=1}^{s} \vec{inp}[\hat{j}] \,\|\, \vec{\pi}[\hat{j}]$
  $\boxed{inp \leftarrow \sum_{\hat{j}=1}^{s} \vec{inp}[\hat{j}];\ \pi \leftarrow \sum_{\hat{j}=1}^{s} \vec{\pi}[\hat{j}]}$
  $\boxed{vf \leftarrow \mathsf{FLP.Query}(inp, \pi, jr; qr)}$
  $d \leftarrow \mathsf{FLP.Decide}(vf)$
  for $\hat{j} \in [s]$:
    $jseed_{\hat{j}} \leftarrow \vec{st}[\hat{j}];\ jseed'_{\hat{j}} \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \vec{rseed})$
    $acc_{\hat{j}} \leftarrow d \wedge [[jseed_{\hat{j}} = jseed'_{\hat{j}}]]$
    $w \leftarrow (w \vee [acc_{\hat{j}} \wedge \mathsf{refineFromShares}(\varepsilon, \vec{x}) \notin \mathcal{L}])$
    $\boxed{w \leftarrow (w \vee [acc_{\hat{j}} \wedge inp \notin \mathcal{L}])}$
  ret $(w, (msg_{\mathrm{Init}}, (\vec{vfs}[\hat{j}], \vec{rseed}[\hat{j}]))_{\hat{j}\in[s]})$
```

Figure 15: Game $\underline{\mathsf{G}}_2$ (left) and game $\underline{\mathsf{G}}_3$ (right) for the proof of Theorem 1.

the same seed. Each query to $\underline{\mathsf{Prep}}$ derives its joint randomness seed from a unique nonce, so duplicate seeds require a collision between two queries to $\underline{\mathsf{RO}}_6$ or between two vectors of hints. Both seeds and hints are randomly sampled by random oracles $\underline{\mathsf{RO}}_6$ and $\underline{\mathsf{RO}}_7$ respectively, so we limit the probability of both types of collision with a birthday bound over the q_{Prep} queries to $\underline{\mathsf{Prep}}$:

$$\frac{{q_{\mathsf{Prep}}}^2}{2^{\kappa+1}} + \frac{{q_{\mathsf{Prep}}}^2}{2^{\kappa \cdot s+1}} < \frac{{q_{\mathsf{Prep}}}^2}{2^{\kappa}} .$$

Since the games are identical until $\texttt{bad}$ gets set, we have

$$\Pr[\underline{\mathsf{G}}_4(B)] \leq \Pr[\underline{\mathsf{G}}_5(B)] + \frac{{q_{\mathsf{Prep}}}^2}{2^{\kappa}} . \tag{6}$$

We are now ready to reduce to FLP soundness. To do so, we construct a malicious prover P^* in Figure 17 from B whose advantage in the FLP soundness experiment is related to B's advantage in winning game $\underline{\mathsf{G}}_5$. Recall from Figure 2 that the prover is called twice, first to choose an input and a second time to generate a proof. The prover is given joint randomness jr in this second call, after committing to the input. Thus, in our reduction we must extract this input from B random oracle queries, then program the random oracle with jr before proceeding.

The malicious prover P^* runs B in a simulation of $\underline{\mathsf{G}}_5$. Its random oracle queries are answered by lazy-evaluating a table Rand; all oracle queries are handled the same way except for a distinguished query, which will be programmed using the jr string generated as part of the malicious prover's experiment. At the start of the simulation, the prover P^* samples $i^* \leftarrow\!\!\$\ [q_1 + q_{\mathsf{Prep}}]$. On the i^* unique invocation of $\underline{\mathsf{RO}}_1$ (see ROExt_1 in Figure 17), the prover checks the table Rand for a nonce n and input shares $inp_1, \ldots, inp_s$ that give rise

```
Game $\underline{\mathsf{G}_3}(B)$ $\boxed{\underline{\mathsf{G}_4}(B)}$:
1 $w \leftarrow \texttt{false};\ sk \leftarrow_{\$} \{0,1\}^{\kappa}$
2 $B^{\underline{\mathsf{RO}},\underline{\mathsf{Prep}}}(\,)$; ret $w$

$\underline{\mathsf{Prep}}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$:
3  if $\mathrm{Used}[n] \neq \bot$: ret $\bot$
4  $\mathrm{Used}[n] \leftarrow \top$
5  for $\hat{j} \in [s]$:
6    $(\vec{inp}[\hat{j}], \vec{\pi}[\hat{j}], blind) \leftarrow \mathsf{Unpack}(\hat{j}, \vec{x}[\hat{j}])$
7    $(\vec{\rho},) \leftarrow \vec{msg}$; $\vec{\rho}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_7(blind, \hat{j} \,\|\, n \,\|\, \vec{inp}[\hat{j}])$
8    $\vec{rseed}[\hat{j}] \leftarrow \vec{\rho}[\hat{j}]$
9    $\vec{st}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_6(0^{\kappa}, \vec{\rho})$ //joint rand seed
10 $jr \leftarrow \underline{\mathsf{RO}}_1(\vec{st}[1], \varepsilon)$
11 $qr \leftarrow \underline{\mathsf{RO}}_5(sk, n)$
12 $inp \leftarrow \sum_{\hat{j}=1}^{s} \vec{inp}[\hat{j}]$; $\pi \leftarrow \sum_{\hat{j}=1}^{s} \vec{\pi}[\hat{j}]$
13 $vf \leftarrow \mathsf{FLP.Query}(inp, \pi, jr; qr)$
14 $\boxed{vf \leftarrow_{\$} \mathsf{FLP.Query}(inp, \pi, jr)}$
15 $d \leftarrow \mathsf{FLP.Decide}(vf)$
16 for $\hat{j} \in [s]$:
17   $jseed_{\hat{j}} \leftarrow \vec{st}[\hat{j}]$; $jseed'_{\hat{j}} \leftarrow \underline{\mathsf{RO}}_6(0^{\kappa}, \vec{rseed})$
18   $acc_{\hat{j}} \leftarrow d \wedge [[jseed_{\hat{j}} = jseed'_{\hat{j}}]]$
19   $w \leftarrow (w \vee [acc_{\hat{j}} \wedge inp \notin \mathcal{L}])$
20 ret $(w, (msg_{\mathrm{Init}}, (\vec{vfs}[\hat{j}], \vec{rseed}[\hat{j}]))_{\hat{j} \in [s]})$
```

```
Game $\underline{\mathsf{G}_4}(B)$ $\boxed{\underline{\mathsf{G}_5}(B)}$:
1 $w \leftarrow \texttt{false};\ sk \leftarrow_{\$} \{0,1\}^{\kappa}$; $\boxed{\mathcal{J} \leftarrow \emptyset}$
2 $B^{\underline{\mathsf{RO}},\underline{\mathsf{Prep}}}(\,)$; ret $w$

$\underline{\mathsf{Prep}}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$:
3  if $\mathrm{Used}[n] \neq \bot$: ret $\bot$
4  $\mathrm{Used}[n] \leftarrow \top$
5  for $\hat{j} \in [s]$:
6    $(\vec{inp}[\hat{j}], \vec{\pi}[\hat{j}], blind) \leftarrow \mathsf{Unpack}(\hat{j}, \vec{x}[\hat{j}])$
7    $(\vec{\rho},) \leftarrow \vec{msg}$; $\vec{\rho}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_7(blind, \hat{j} \,\|\, n \,\|\, \vec{inp}[\hat{j}])$
8    $\vec{rseed}[\hat{j}] \leftarrow \vec{\rho}[\hat{j}]$
9    $\vec{st}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_6(0^{\kappa}, \vec{\rho})$ //joint rand seed
10 $\boxed{\text{if } \vec{st}[1] \in \mathcal{J}\text{: } \texttt{bad} \leftarrow \texttt{true}}$
11 $\boxed{\mathcal{J} \leftarrow \mathcal{J} \cup \{\vec{st}[1]\}}$
12 $jr \leftarrow \underline{\mathsf{RO}}_1(\vec{st}[1], \varepsilon)$
13 $inp \leftarrow \sum_{\hat{j}=1}^{s} \vec{inp}[\hat{j}]$; $\pi \leftarrow \sum_{\hat{j}=1}^{s} \vec{\pi}[\hat{j}]$
14 $vf \leftarrow_{\$} \mathsf{FLP.Query}(inp, \pi, jr)$
15 $d \leftarrow \mathsf{FLP.Decide}(vf)$
16 for $\hat{j} \in [s]$:
17   $jseed_{\hat{j}} \leftarrow \vec{st}[\hat{j}]$; $jseed'_{\hat{j}} \leftarrow \underline{\mathsf{RO}}_6(0^{\kappa}, \vec{rseed})$
18   $acc_{\hat{j}} \leftarrow d \wedge [[jseed_{\hat{j}} = jseed'_{\hat{j}}]]$
19   $w \leftarrow (w \vee [\ \boxed{\neg\texttt{bad} \wedge}\ acc_{\hat{j}} \wedge inp \notin \mathcal{L}])$
20 ret $(w, (msg_{\mathrm{Init}}, (\vec{vfs}[\hat{j}], \vec{rseed}[\hat{j}]))_{\hat{j} \in [s]})$
```

Figure 16: Fourth and fifth intermediate games for the proof of Theorem 1.

to the seed *jseed* provided as input. If successful, the prover outputs $inp_1 + \cdots + inp_s$ as its challenge input, awaits the response jr, and sets $\mathrm{Rand}[1, jseed, \varepsilon] \leftarrow jr$ (line 11). It also records $n^* \leftarrow n$ for use later on.

The simulation of $\underline{\mathsf{Prep}}$ queries is identical except after two events. First, the prover P^* halts and concedes if two $\underline{\mathsf{Prep}}$ queries generate the same joint randomness seed $\vec{st}[1]$. (Adversary B loses in this case.) Second, if $n = n^*$, then P^* immediately halts and outputs the proof π computed on line 30. If the simulation has reached this point, then the probability that P^* wins its game is at least the probability that the game sets $w \leftarrow \texttt{true}$ on line 36. Conditioning on the probability that P^* guesses the winning query to $\underline{\mathsf{RO}}_1$, we have that

$$\Pr\left[\,\underline{\mathsf{G}_5}(B)\,\right] \leq (q_1 + q_{\mathsf{Prep}}) \cdot \epsilon\,. \tag{7}$$

The claimed bound follows by gathering up all of the bounds across the games and simplifying. □

C.2 Prio3 Privacy (Theorem 2)

We begin by instantiating the privacy game $\mathsf{Exp}^{\mathrm{priv}}_{\Pi,t}$ for Prio3 VDAF Π. Game $\underline{\mathsf{G}_0}$ in Figure 18 was constructed by inlining Π's constituent algorithms and cleaning up the control flow. In addition, calls to RG_i have been substituted with calls to a random oracle $\underline{\mathsf{RO}}_i$. Let q_i denote the number of queries A makes to $\underline{\mathsf{RO}}_i$; note that $q_{\mathsf{RG}} = q_1 + \cdots + q_7$.

In our first game hop, we modify $\underline{\mathsf{Shard}}$ oracle's behavior after setting flag $\texttt{bad}_1$ on line 7. In the new game, $\underline{\mathsf{G}_1}$ (Figure 18), the nonce n is sampled without replacement, ensuring that each nonce is used is unique. Applying the Fundamental Lemma of Game Playing [14], and using a birthday bound for the probability of $\texttt{bad}_1$ getting set,

$$\Pr\left[\,\underline{\mathsf{G}_0}(A)\,\right] \leq \Pr\left[\,\underline{\mathsf{G}_1}(A)\,\right] + \frac{{q_{\mathsf{Shard}}}^2}{|\mathcal{N}|}\,. \tag{8}$$

```
Adversary $P^*[B]()$:
1  $w \gets \texttt{false}$; $sk \gets_{\$} \{0,1\}^\kappa$; $\texttt{bad} \gets \texttt{false}$; $\mathcal{J} \gets \emptyset$
2  $ctr \gets 0$; $n^* \gets \bot$; $i^* \gets_{\$} [q_1 + q_{\mathsf{Prep}}]$
3  $B^{\mathsf{ROExt}_1, \underline{\mathsf{RO}}_2, \ldots, \underline{\mathsf{RO}}_7 \mathsf{PrepSim}}()$; ret $w$

$\mathsf{ROExt}_1(seed, cntxt)$:
4  if $\mathrm{Rand}[1, seed, cntxt] \neq \bot$: ret $\underline{\mathsf{RO}}_1(seed, cntxt)$
5  $ctr \gets ctr + 1$
6  if $ctr = i^* \wedge (\exists\, n, (blind_{\hat{j}}, inp_{\hat{j}}, \rho_{\hat{j}})_{\hat{j} \in [s]})$
7      $\left(\forall \hat{j}\right)$ $\mathrm{Rand}[7, blind_{\hat{j}}, \hat{j} \parallel n \parallel inp_{\hat{j}}] = \rho_{\hat{j}}$
8      $\wedge\, \mathrm{Rand}[6, 0^\kappa, (\rho_1, \ldots, \rho_s)] = seed$:
9    output $inp_1 + \cdots + inp_s$ and wait for $jr$.
10   $n^* \gets n$; $\mathrm{Rand}[1, seed, cntxt] \gets jr$
11 ret $\underline{\mathsf{RO}}_1(seed, cntxt)$

$\underline{\mathsf{RO}}_i(seed, cntxt)$:
12 $l \gets (jl, n, m, pl, ql)$
13 if $\mathrm{Rand}[i, seed, cntxt] = \bot$:
14   if $i \leq 5$: $\mathrm{Rand}[i, seed, cntxt] \gets_{\$} \mathbb{F}^{l[i]}$
15   else: $\mathrm{Rand}[i, seed, cntxt] \gets_{\$} \{0,1\}^\kappa$
16 ret $\mathrm{Rand}[i, seed, cntxt]$

$\mathsf{PrepSim}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$:
17 if $\mathrm{Used}[n] \neq \bot$: ret $\bot$
18 $\mathrm{Used}[n] \gets \top$
19 for $\hat{j} \in [s]$:
20   $(\vec{inp}[\hat{j}], \vec{\pi}[\hat{j}], blind) \gets \mathsf{Unpack}(\hat{j}, \vec{x}[\hat{j}])$
21   $(\vec{\rho}, ) \gets \vec{msg}$; $\vec{\rho}[\hat{j}] \gets \underline{\mathsf{RO}}_7(blind, \hat{j} \parallel n \parallel \vec{inp}[\hat{j}])$
22   $\vec{rseed}[\hat{j}] \gets \vec{\rho}[\hat{j}]$
23   $\vec{st}[\hat{j}] \gets \underline{\mathsf{RO}}_6(0^\kappa, \vec{\rho})$ //joint rand seed
24 if $\vec{st}[1] \in \mathcal{J}$: $\texttt{bad} \gets \texttt{true}$; halt.
25 $\mathcal{J} \gets \mathcal{J} \cup \{\vec{st}[1]\}$
26 $jr \gets \mathsf{ROExt}_1(\vec{st}[1], \varepsilon)$
27 $inp \gets \sum_{\hat{j}=1}^{s} \vec{inp}[\hat{j}]$; $\pi \gets \sum_{\hat{j}=1}^{s} \vec{\pi}[\hat{j}]$
28 if $n = n^*$: output $\pi$ and halt.
29 $vf \gets_{\$} \mathsf{FLP.Query}(inp, \pi, jr)$
30 $d \gets \mathsf{FLP.Decide}(vf)$
31 for $\hat{j} \in [s]$:
32   $jseed_{\hat{j}} \gets \vec{st}[\hat{j}]$; $jseed'_{\hat{j}} \gets \underline{\mathsf{RO}}_6(0^\kappa, \vec{rseed})$
33   $acc_{\hat{j}} \gets d \wedge [[jseed_{\hat{j}} = jseed'_{\hat{j}}]]$
34   $w \gets (w \vee [\neg\texttt{bad} \wedge acc_{\hat{j}} \wedge inp \notin \mathcal{L}])$
35 ret $(w, (msg_{\mathrm{Init}}, (\vec{vfs}[\hat{j}], \vec{rseed}[\hat{j}]))_{\hat{j} \in [s]})$
```

Figure 17: Malicious prover P^* for the proof of Theorem 1. The lookup in the random oracle table Rand on lines 6–8 can performed efficiently by creating a reverse-lookup table; we omit the details for brevity.

Next we replace the adversary A with one that controls all but one aggregator. We construct such an adversary B as a wrapper around A, and show that B wins with at least the probability of A. The adversary B, defined in Figure 19, presents four oracles $\mathsf{ShardSim}$, $\mathsf{SetupSim}$, $\mathsf{PrepSim}$, and AggSim to adversary A, each emulating an oracle in game $\underline{\mathsf{G}}_1$. Algorithms $\mathsf{SetupSim}$, $\mathsf{PrepSim}$, AggSim are computed by B just as the respective oracles in game $\underline{\mathsf{G}}_1$ except that queries pertaining to aggregator z are forwarded to B's own oracles. Algorithm $\mathsf{ShardSim}$ forwards A's query to $\underline{\mathsf{Shard}}$ in the natural way, but returns the shares of the aggregators deemed honest by A.

We claim that B perfectly simulates $\underline{\mathsf{G}}_1(A)$. This is obvious for $\mathsf{ShardSim}$ and for queries for which $\hat{j} = z$; in these cases, B simply forwards its queries to the appropriate oracles without changing their inputs. The only difference is that $\underline{\mathsf{Shard}}$ returns more input shares than A requests; B stores these extra input shares for its own use and does not reveal them. By construction, this is a subset of the input shares returned by the query.

When A makes queries to $\mathsf{SetupSim}$, $\mathsf{PrepSim}$, or AggSim with $\hat{j} \in \mathcal{V} \backslash \{z\}$, our wrapper adversary performs the operations of $\underline{\mathsf{Setup}}$, $\underline{\mathsf{Prep}}$, or $\underline{\mathsf{Agg}}$ respectively. Effectively, adversary B uses its stored input shares to fill in entries of tables In, Batch, Setup, Status, St, and Out exactly as the real privacy game would. Since each entry is disambiguated by its $\hat{j}$, there is no overlap with the tables maintained by the game; every table entry read by B must first have been written by B and thus all the information it needs to simulate the game perfectly is accessible. It follows that

$$\Pr\left[\, \underline{\mathsf{G}}_1(A) \,\right] = \Pr\left[\, \underline{\mathsf{G}}_1(B) \,\right]. \tag{9}$$

In the next game hop (Figure 20) we make some simplifying changes, including cleaning up the $\texttt{bad}_1$ flag and substituting $\{z\}$ for $\mathcal{V}$ and simplifying accordingly. (We do not highlight this change in Figure 20, as it is fairly straightforward.) We also make the following breaking change: In game $\underline{\mathsf{G}}_2$, we program the table Rand with values chosen by the $\underline{\mathsf{Shard}}$ oracle for the joint randomness, prover randomness, and query randomness. Accordingly, we pass these joint randomness and query randomness to the honest aggregator via its input share ($\mathrm{In}[\hat{k}, z]$; see line 29). This is to simplify bookkeeping in the next step.

Game $\underline{\mathsf{G}}_2$ is identical to game $\underline{\mathsf{G}}_1$ until programming Rand overwrites an already existing value on line 18, 19, 20, or 21.

```
Game $\underline{\mathsf{G}}_0(A)$ $\boxed{\underline{\mathsf{G}}_1(A)}$:
  $(st_A, \mathcal{V}, (sk_{\hat{j}})_{\hat{j}\in\mathcal{V}}) \leftarrow\!\!\$\; A^{\underline{\mathsf{RO}}}(\,)$; $\mathcal{T} \leftarrow [s] \setminus \mathcal{V}$
  if $|\mathcal{V}| + t \neq s$ return $\perp$
  $b \leftarrow\!\!\$\; \{0,1\}$; $b' \leftarrow\!\!\$\; A^{\underline{\mathsf{RO}},\underline{\mathsf{Shard}},\underline{\mathsf{Setup}},\underline{\mathsf{Prep}},\underline{\mathsf{Agg}}}(st_A)$
  ret $b = b'$

$\underline{\mathsf{Shard}}(\hat{k} \in \mathbb{N}, m_0, m_1 \in \mathcal{I})$:
  if $\mathrm{Used}[\hat{k}] \neq \perp$: ret $\perp$
  $n \leftarrow\!\!\$\; \mathcal{N}$
  if $n \in \mathcal{N}^*$: $\mathtt{bad}_1 \leftarrow \mathtt{true}$; $\boxed{n \leftarrow\!\!\$\; \mathcal{N} \setminus \mathcal{N}^*}$
  $\mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}$
  $inp \leftarrow \mathsf{Encode}(m_b)$
  for $\hat{j} \in [2..s]$:
    $blind_{\hat{j}}, xseed_{\hat{j}}, pseed_{\hat{j}} \leftarrow\!\!\$\; \{0,1\}^\kappa$
    $\vec{x}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_2(xseed_{\hat{j}}, \hat{j})$
    $\vec{rseed}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_7(blind_{\hat{j}}, \hat{j} \,\|\, n \,\|\, \vec{x}[\hat{j}])$
  $\vec{x}[1] \leftarrow inp - \sum_{\hat{j}=2}^{s} \vec{x}[\hat{j}]$
  $blind_1 \leftarrow\!\!\$\; \{0,1\}^\kappa$; $ps \leftarrow\!\!\$\; \{0,1\}^\kappa$
  $\vec{rseed}[1] \leftarrow \underline{\mathsf{RO}}_7(blind_1, 1 \,\|\, n \,\|\, \vec{x}[1])$
  $jseed \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \vec{rseed})$; $jr \leftarrow \underline{\mathsf{RO}}_1(jseed, \varepsilon)$
  $pr \leftarrow \underline{\mathsf{RO}}_4(ps, \varepsilon)$
  $\vec{\pi}[1] \leftarrow \mathsf{Prove}(inp, jr\;\; pr)$
  $\vec{\pi}[1] \leftarrow \vec{\pi}[1] - \sum_{\hat{j}=2}^{s} \underline{\mathsf{RO}}_3(pseed_{\hat{j}}, \hat{j})$
  $\vec{x}[1] \leftarrow (\vec{x}[1], \vec{\pi}[1], blind_1)$
  for $\hat{j} \in [2..s]$:
    $\vec{x}[\hat{j}] \leftarrow (xseed_{\hat{j}}, pseed_{\hat{j}}, blind_{\hat{j}})$
  $\mathrm{Pub}[\hat{k}] \leftarrow \vec{rseed}$; $\mathrm{In}[\hat{k}, \cdot] \leftarrow \vec{x}$
  $\mathrm{Used}[\hat{k}] \leftarrow (n, m_0, m_1)$
  ret $(n, \mathrm{Pub}[\hat{k}], (\mathrm{In}[\hat{k}, \hat{j}])_{\hat{j}\in\mathcal{T}})$

$\underline{\mathsf{Setup}}(\hat{i} \in \mathbb{N}, \hat{j} \in \mathcal{V}, st_{\mathrm{Init}} \in \{\varepsilon\})$:
  if $\mathrm{Status}[\hat{i}, \hat{j}] \neq \perp$ or $|\mathrm{Setup}[\cdot, \hat{j}]| > 0$: ret $\perp$
  $\mathrm{Setup}[\hat{i}, \hat{j}] \leftarrow st_{\mathrm{Init}}$
  $\mathrm{Status}[\hat{i}, \hat{j}] \leftarrow \mathtt{running}$

$\underline{\mathsf{Prep}}(\hat{i} \in \mathbb{N}, \hat{j} \in \mathcal{V}, \hat{k} \in \mathbb{N}, \vec{msg} \in \mathcal{M}^*)$:
  if $\mathrm{Status}[\hat{i}, \hat{j}] \neq \mathtt{running}$ or $\mathrm{In}[\hat{k}, \hat{j}] = \perp$:
    ret $\perp$
  if $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] = \perp$:
    $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow \mathrm{Setup}[\hat{i}, \hat{j}]$
    $\vec{msg} \leftarrow (\mathrm{Pub}[\hat{k}], )$
  $(n, m_0, m_1) \leftarrow \mathrm{Used}[\hat{k}]$
  if $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] = \varepsilon$: // Process initial message from client
    $(inp, \pi, blind) \leftarrow \mathsf{Unpack}(\hat{j}, \mathrm{In}[\hat{k}, \hat{j}])$
    $(\vec{rseed}, ) \leftarrow \vec{msg}$
    $\vec{rseed}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_7(blind, \hat{j} \,\|\, n \,\|\, inp)$
    $jseed \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \vec{rseed})$; $jr \leftarrow \underline{\mathsf{RO}}_1(jseed, \varepsilon)$
    $qr \leftarrow \underline{\mathsf{RO}}_5(sk_{\hat{j}}, n)$
    $msg \leftarrow (\mathsf{Query}(inp, \pi, jr; qr), \vec{rseed}[\hat{j}])$
    $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow (jseed, \mathsf{Truncate}(inp))$
    ret $(\mathtt{running}, msg)$
  // Process broadcast messages from aggregators
  $(jseed, y) \leftarrow \mathrm{St}[\hat{i}, \hat{j}, \hat{k}]$
  $(\vec{vfs}[\hat{j}], \vec{rseed}[\hat{j}])_{\hat{j}\in[s]} \leftarrow \vec{msg}$
  $acc \leftarrow \mathsf{Decide}(\sum_{\hat{j}=1}^{s} \vec{vfs}[\hat{j}])$
  $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow \perp$
  if $acc = 0$ or $jseed \neq \underline{\mathsf{RO}}_6(0^\kappa, \vec{rseed})$:
    ret $(\mathtt{failed}, \perp)$
  $\mathrm{Out}[\hat{i}, \hat{j}, \hat{k}] \leftarrow y$
  $\mathrm{Batch}_0[\hat{i}, \hat{j}, \hat{k}] \leftarrow m_0$
  $\mathrm{Batch}_1[\hat{i}, \hat{j}, \hat{k}] \leftarrow m_1$
  ret $(\mathtt{finished}, \perp)$

$\underline{\mathsf{Agg}}(\hat{i} \in \mathbb{N}, \hat{j} \in \mathcal{V})$:
  if $\mathrm{Status}[\hat{i}, \hat{j}] \neq \mathtt{running}$: ret $\perp$
  if $F(\mathrm{Batch}_0[\hat{i}, \hat{j}, \cdot]) \neq F(\mathrm{Batch}_1[\hat{i}, \hat{j}, \cdot])$
     and $(\forall j, j' \in \mathcal{V})\; sk_j = sk_{j'}$: ret $\perp$
  $\mathrm{Status}[\hat{i}, \hat{j}] \leftarrow \mathtt{finished}$
  $\vec{y} \leftarrow \mathrm{Out}[\hat{i}, \hat{j}, \cdot]$
  ret $\sum_{i=1}^{|\vec{y}|} \vec{y}[i]$

$\underline{\mathsf{RO}}_i(seed, cntxt)$:
  $l \leftarrow (jl, n, m, pl, ql)$
  if $\mathrm{Rand}[i, seed, cntxt] = \perp$:
    if $i \leq 5$: $\mathrm{Rand}[i, seed, cntxt] \leftarrow\!\!\$\; \mathbb{F}^{l[i]}$
    else: $\mathrm{Rand}[i, seed, cntxt] \leftarrow\!\!\$\; \{0,1\}^\kappa$
  ret $\mathrm{Rand}[i, seed, cntxt]$
```

Figure 18: Games $\underline{\mathsf{G}}_0$ and $\underline{\mathsf{G}}_1$ for the proof of Theorem 2. This game is identical to the privacy game for Π, except the Shard, Prep, and Agg algorithms have been inlined. Algorithm Unpack is as defined in Figure 5. The random oracles $\underline{\mathsf{RO}}_i$ are lazy-evaluated in a table Rand.

```
Adversary $B^{\underline{\mathsf{RO}}}[A]()$:
  $(st_A, \mathcal{V}, (sk_{\hat{j}})_{\hat{j} \in \mathcal{V}}) \leftarrow\!\!\$\; A^{\underline{\mathsf{RO}}}()$
  $z \leftarrow\!\!\$\; \mathcal{V}$; $\mathcal{V}' \leftarrow \{z\}$; $\mathcal{T} \leftarrow [s] \setminus \mathcal{V}$
  $st_B \leftarrow (st_A, z, \mathcal{T}, \mathcal{V}, (sk_{\hat{j}})_{\hat{j} \in \mathcal{V}})$
  ret $(st_B, \mathcal{V}', (sk_z))$

Adversary $B^{\underline{\mathsf{RO}}, \underline{\mathsf{Shard}}, \underline{\mathsf{Setup}}, \underline{\mathsf{Prep}}, \underline{\mathsf{Agg}}}[A](st_B)$:
  $(st_A, z, \mathcal{T}, \mathcal{V}, (sk_{\hat{j}})_{\hat{j} \in \mathcal{V}}) \leftarrow st_B$
  $b' \leftarrow\!\!\$\; A^{\underline{\mathsf{RO}}, \mathsf{ShardSim}, \mathsf{SetupSim}, \mathsf{PrepSim}, \mathsf{AggSim}}(st_A)$
  ret $b'$
```

Figure 19: Wrapper adversary B for the proof of Theorem 2. Algorithms $\mathsf{SetupSim}$, $\mathsf{PrepSim}$, AggSim are evaluated by B just as the respective oracles in game $\underline{\mathsf{G}}_0$ except that queries pertaining to aggregator z are forwarded to B's own oracles. Algorithm $\mathsf{ShardSim}$ forwards A's query to $\underline{\mathsf{Shard}}$ in the natural way, but returns the shares of the aggregators deemed honest by A.

- Line 18: Adversary B either has to guess $jseed$ or guess the input to $\underline{\mathsf{RO}}_6$ used to derive it. For the latter it must must guess $\vec{rseed}$ or all of the corresponding inputs to $\underline{\mathsf{RO}}_7$, which include the blinds generated by oracle $\underline{\mathsf{Shard}}$. Taking union bound over all the queries to $\underline{\mathsf{Shard}}$, the game overwrites Rand at this point with probability at most $q_1 q_{\mathsf{Shard}}/2^\kappa + (q_6 + q_7) q_{\mathsf{Shard}}/(2^{s \cdot \kappa})$.
- Line 19: B must guess the ps generated by oracle $\underline{\mathsf{Shard}}$, so the game overwrites the table with probability at most $q_4 q_{\mathsf{Shard}}/2^\kappa$.
- Line 20: B must guess the nonce n generated by the oracle. The game overwrites the table here with probability at most $q_5 q_{\mathsf{Shard}}/|\mathcal{N}|$.
- Line 21: B must guess $\vec{rseed}$ or all of the corresponding inputs to $\underline{\mathsf{RO}}_7$, so the game overwrites the table with probability at most $(q_6 + q_7) q_{\mathsf{Shard}}/(2^{s \cdot \kappa})$.

We bound the probability of B distinguishing between these games by the probability that any one of these events occurs, Gathering up the terms yields

$$\Pr\left[\, \underline{\mathsf{G}}_1(B) \,\right] \le \Pr\left[\, \underline{\mathsf{G}}_2(B) \,\right] \tag{10}$$

$$+ \frac{(q_1 + q_4) q_{\mathsf{Shard}}}{2^\kappa} + \frac{(q_6 + q_7) q_{\mathsf{Shard}}}{2^{s \cdot \kappa - 1}} + \frac{q_5 q_{\mathsf{Shard}}}{|\mathcal{N}|} \,. \tag{11}$$

In the next game hop (see $\underline{\mathsf{G}}_3$ in the left panel of Figure 21) we prepare to ensure that all of the input shares $\vec{x}$, proof shares $\vec{\pi}$, and the public share $\vec{rseed}$ sampled by the $\underline{\mathsf{Shard}}$ oracle are uniform random. We do so by sampling these values prior to processing m_b and programming the random oracle with the sample values (lines 3–12) so long as doing so does overwrite existing values (see procedure PO on lines 37–40). The game sets a flag $\mathtt{bad}_4$ if $\underline{\mathsf{Shard}}$ would have overwritten an existing value. This does not change the adversary's view of the experiment, so

$$\Pr\left[\, \underline{\mathsf{G}}_2(B) \,\right] = \Pr\left[\, \underline{\mathsf{G}}_3(B) \,\right] . \tag{12}$$

Next, in game $\underline{\mathsf{G}}_4$ (top-right panel of Figure 21) we change oracle $\underline{\mathsf{Shard}}$'s behavior after $\mathtt{bad}_4$ gets set. In particular, if ever PO is called on an input $(i, seed, cntxt, out)$ for which Rand$[i, seed, cntxt]$, the value is overwritten. Game $\underline{\mathsf{G}}_4$ is identical to game $\underline{\mathsf{G}}_3$ until $\mathtt{bad}_4$ gets set. Then we apply the Fundamental Lemma of Game Playing [14] to show that

$$\Pr\left[\, \underline{\mathsf{G}}_3(B) \,\right] \le \Pr\left[\, \underline{\mathsf{G}}_4(B) \,\right] + \Pr\left[\, \underline{\mathsf{G}}_4(B) \text{ sets } \mathtt{bad}_4 \,\right] \tag{13}$$

$$\le \Pr\left[\, \underline{\mathsf{G}}_4(B) \,\right] + \frac{((s-1)(q_2 + q_3) + s(q_7)) q_{\mathsf{Shard}}}{2^\kappa} \,. \tag{14}$$

The probability that B sets the $\mathtt{bad}_4$ flag in Game $\underline{\mathsf{G}}_4$ is the probability that B makes a random oracle query that gets overwritten on line 9, 10, 11, or 12. On each line, the random oracle is programmed with a uniform random string sampled by the oracle prior to being revealed to the adversary. Rolling out the for-loop on line 8 and taking a union bound over all $\underline{\mathsf{Shard}}$ queries yields the claimed bound.

```
Game $\underline{\mathsf{G}}_1(B)$ $\boxed{\underline{\mathsf{G}}_2(B)}$:
$(st_B, \{z\}, (sk_z,)) \leftarrow_{\$} B^{\underline{\mathsf{RO}}}()$; $\mathcal{T} \leftarrow [s] \setminus \{z\}$
$b \leftarrow_{\$} \{0,1\}$; $b' \leftarrow_{\$} B^{\underline{\mathsf{RO}},\underline{\mathsf{Shard}},\underline{\mathsf{Setup}},\underline{\mathsf{Prep}},\underline{\mathsf{Agg}}}(st_B)$
ret $b = b'$

$\underline{\mathsf{Shard}}(\hat{k} \in \mathbb{N}, m_0, m_1 \in \mathcal{I})$:
if $\mathrm{Used}[\hat{k}] \neq \bot$: ret $\bot$
$n \leftarrow_{\$} \mathcal{N} \setminus \mathcal{N}^*$; $\mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}$
$inp \leftarrow \mathsf{Encode}(m_b)$
for $\hat{j} \in [2..s]$:
    $blind_{\hat{j}}, xseed_{\hat{j}}, pseed_{\hat{j}} \leftarrow_{\$} \{0,1\}^\kappa$
    $\vec{x}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_2(xseed_{\hat{j}}, \hat{j})$
    $\vec{rseed}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_7(blind_{\hat{j}}, \hat{j} \parallel n \parallel \vec{x}[\hat{j}])$
$\vec{x}[1] \leftarrow inp - \sum_{\hat{j}=2}^{s} \vec{x}[\hat{j}]$
$blind_1 \leftarrow_{\$} \{0,1\}^\kappa$; $ps \leftarrow_{\$} \{0,1\}^\kappa$
$\vec{rseed}[1] \leftarrow \underline{\mathsf{RO}}_7(blind_1, 1 \parallel n \parallel \vec{x}[1])$
$jseed \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \vec{rseed})$; $jr \leftarrow \underline{\mathsf{RO}}_1(jseed, \varepsilon)$
$pr \leftarrow \underline{\mathsf{RO}}_4(ps, \varepsilon)$
$jseed \leftarrow_{\$} \{0,1\}^\kappa$
$jr \leftarrow_{\$} \mathbb{F}^{jl}$; $pr \leftarrow_{\$} \mathbb{F}^{pl}$; $qr \leftarrow_{\$} \mathbb{F}^{ql}$
$\mathrm{Rand}[1, jseed, \varepsilon] \leftarrow jr$
$\mathrm{Rand}[4, ps, \varepsilon] \leftarrow pr$
$\mathrm{Rand}[5, sk_z, n] \leftarrow qr$
$\mathrm{Rand}[6, 0^\kappa, \vec{rseed}] \leftarrow jseed$
$\vec{\pi}[1] \leftarrow \mathsf{Prove}(inp, jr\,; pr)$
$\vec{\pi}[1] \leftarrow \vec{\pi}[1] - \sum_{\hat{j}=2}^{s} \underline{\mathsf{RO}}_3(pseed_{\hat{j}}, \hat{j})$
$\vec{x}[1] \leftarrow (\vec{x}[1], \vec{\pi}[1], blind_1)$
for $\hat{j} \in [2..s]$:
    $\vec{x}[\hat{j}] \leftarrow (xseed_{\hat{j}}, pseed_{\hat{j}}, blind_{\hat{j}})$
$\mathrm{Pub}[\hat{k}] \leftarrow \vec{rseed}$
$\mathrm{In}[\hat{k}, \cdot] \leftarrow \vec{x}$
$\mathrm{In}[\hat{k}, z] \leftarrow (\vec{x}[z], jseed, jr, qr)$
$\mathrm{Used}[\hat{k}] \leftarrow (n, m_0, m_1)$
ret $(n, \mathrm{Pub}[\hat{k}], (\mathrm{In}[\hat{k}, \hat{j}])_{\hat{j} \in \mathcal{T}})$

$\underline{\mathsf{Setup}}(\hat{i} \in \mathbb{N}, \hat{j} \in \{z\}, st_{\mathrm{Init}} \in \{\varepsilon\})$:
if $\mathrm{Status}[\hat{i}, \hat{j}] \neq \bot$ or $|\mathrm{Setup}[\cdot, \hat{j}]| > 0$: ret $\bot$
$\mathrm{Setup}[\hat{i}, \hat{j}] \leftarrow st_{\mathrm{Init}}$
$\mathrm{Status}[\hat{i}, \hat{j}] \leftarrow \mathtt{running}$

$\underline{\mathsf{Prep}}(\hat{i} \in \mathbb{N}, \hat{j} \in \{z\}, \hat{k} \in \mathbb{N}, \vec{msg} \in \mathcal{M}^*)$:
if $\mathrm{Status}[\hat{i}, \hat{j}] \neq \mathtt{running}$ or $\mathrm{In}[\hat{k}, \hat{j}] = \bot$:
    ret $\bot$
if $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] = \bot$:
    $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow \mathrm{Setup}[\hat{i}, \hat{j}]$
    $\vec{msg} \leftarrow (\mathrm{Pub}[\hat{k}],)$
$(n, m_0, m_1) \leftarrow \mathrm{Used}[\hat{k}]$
if $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] = \varepsilon$:  // Process initial message from client
    $(inp, \pi, blind) \leftarrow \mathsf{Unpack}(\hat{j}, \mathrm{In}[\hat{k}, \hat{j}])$
    $(x, jseed, jr, qr) \leftarrow \mathrm{In}[\hat{k}, \hat{j}]$
    $(inp, \pi, blind) \leftarrow \mathsf{Unpack}(\hat{j}, x)$
    $(\vec{rseed},) \leftarrow \vec{msg}$
    $\vec{rseed}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_7(blind, \hat{j} \parallel n \parallel inp)$
    $jseed \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \vec{rseed})$; $jr \leftarrow \underline{\mathsf{RO}}_1(jseed, \varepsilon)$
    $qr \leftarrow \underline{\mathsf{RO}}_5(sk_z, n)$
    $msg \leftarrow (\mathsf{Query}(inp, \pi, jr; qr), \vec{rseed}[\hat{j}])$
    $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow (jseed, \mathsf{Truncate}(inp))$
    ret $(\mathtt{running}, msg)$
// Process broadcast messages from aggregators
$(jseed, y) \leftarrow \mathrm{St}[\hat{i}, \hat{j}, \hat{k}]$
$(\vec{vfs}[\hat{j}], \vec{rseed}[\hat{j}])_{\hat{j} \in [s]} \leftarrow \vec{msg}$
$acc \leftarrow \mathsf{Decide}(\sum_{\hat{j}=1}^{s} \vec{vfs}[\hat{j}])$
$\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow \bot$
if $acc = 0$ or $jseed \neq \underline{\mathsf{RO}}_6(0^\kappa, \vec{rseed})$:
    ret $(\mathtt{failed}, \bot)$
$\mathrm{Out}[\hat{i}, \hat{j}, \hat{k}] \leftarrow y$
$\mathrm{Batch}_0[\hat{i}, \hat{j}, \hat{k}] \leftarrow m_0$
$\mathrm{Batch}_1[\hat{i}, \hat{j}, \hat{k}] \leftarrow m_1$
ret $(\mathtt{finished}, \bot)$

$\underline{\mathsf{Agg}}(\hat{i} \in \mathbb{N}, \hat{j} \in \{z\})$:
if $\mathrm{Status}[\hat{i}, \hat{j}] \neq \mathtt{running}$: ret $\bot$
if $F(\mathrm{Batch}_0[\hat{i}, \hat{j}, \cdot]) \neq F(\mathrm{Batch}_1[\hat{i}, \hat{j}, \cdot])$: ret $\bot$
$\mathrm{Status}[\hat{i}, \hat{j}] \leftarrow \mathtt{finished}$
$\vec{y} \leftarrow \mathrm{Out}[\hat{i}, \hat{j}, \cdot]$
ret $\sum_{i=1}^{|\vec{y}|} \vec{y}[i]$

$\underline{\mathsf{RO}}_i(seed, cntxt)$:
$l \leftarrow (jl, n, m, pl, ql)$
if $\mathrm{Rand}[i, seed, cntxt] = \bot$:
    if $i \leq 5$: $\mathrm{Rand}[i, seed, cntxt] \leftarrow_{\$} \mathbb{F}^{l[i]}$
    else: $\mathrm{Rand}[i, seed, cntxt] \leftarrow_{\$} \{0,1\}^\kappa$
ret $\mathrm{Rand}[i, seed, cntxt]$
```

Figure 20: Game $\underline{\mathsf{G}}_2$ for the proof of Theorem 2.

Next, in game $\underline{\mathsf{G}}_5$ (bottom-right panel of Figure 21) we simplify the $\underline{\mathsf{Shard}}$ oracle by inlining calls to PO and replacing invocations of $\underline{\mathsf{RO}}$ with corresponding value generated by the oracle. These changes do not change the view of the adversary, so

$$\Pr\left[\,\underline{\mathsf{G}}_4(B)\,\right] = \Pr\left[\,\underline{\mathsf{G}}_5(B)\,\right]. \tag{15}$$

Up to this point, we have constructed the "leader" input and proof shares differently than all of the other shares: we pick all other shares randomly, then set $\vec{x}[1] = \mathit{inp} - \sum_{\hat{j}=2}^{s} \vec{x}[\hat{j}]$ and $\vec{\pi}[1] = \pi - \sum_{\hat{j}=2}^{s} \vec{\pi}[\hat{j}]$. In our next game, we instead sample the "leader" shares randomly and compute the shares of the honest aggregator z in a distinguished manner: $\vec{x}[z] = \mathit{inp} - \sum_{\hat{j}\in\mathcal{T}} \vec{x}[\hat{j}]$ and $\vec{\pi}[z] = \pi - \sum_{\hat{j}\in\mathcal{T}} \vec{x}[\hat{j}]$, where $\mathcal{T} = [s]\backslash\{z\}$. If $z = 1$, this changes nothing. Otherwise, consider that in $\underline{\mathsf{G}}_5$, we have

$$\vec{x}[1] = \mathit{inp} - \sum_{\hat{j}=2}^{s} \vec{x}[\hat{j}] = \mathit{inp} - \left(\sum_{\hat{j}\in\mathcal{T}} \vec{x}[\hat{j}] - \vec{x}[z] + \vec{x}[1] \right).$$

If we add $\vec{x}[z] - \vec{x}[1]$ to both sides of this equation, we can see that in $\underline{\mathsf{G}}_4$, it was already true that $\vec{x}[z] = \mathit{inp} - \sum_{\hat{j}\in\mathcal{T}} \vec{x}[\hat{j}]$. The same holds true for $\vec{\pi}[z]$ by an analogous argument. Therefore the distributions of aggregators' 1 and z's input and proof shares are unchanged between $\underline{\mathsf{G}}_4$ and $\underline{\mathsf{G}}_5$, and we have

$$\Pr[\underline{\mathsf{G}}_5(B)] = \Pr[\underline{\mathsf{G}}_6(B)]. \tag{16}$$

In our next game ($\underline{\mathsf{G}}_7$, defined in the right panel of Figure 22) we run the query algorithm for aggregator z in the $\underline{\mathsf{Shard}}$ oracle and only send the result to $\underline{\mathsf{Prep}}$. The adversary cannot detect the timing of when this algorithm is run, so we have

$$\Pr[\underline{\mathsf{G}}_6(B)] = \Pr[\underline{\mathsf{G}}_7(B)]. \tag{17}$$

In the next game ($\underline{\mathsf{G}}_8$, defined in the left panel of Figure 23) we run $\mathsf{View}_{\mathsf{FLP}}$ (as defined in Section 2) on input inp to get jr, qr, and a verifier σ and use these to compute $\underline{\mathsf{Shard}}$'s output. We have defined $\vec{x}[z] = \mathit{inp} - \sum_{\hat{j}\in\mathcal{T}} \vec{x}[\hat{j}]$ and $\vec{x}[z] = \pi - \sum_{\hat{j}\in\mathcal{T}} \vec{\pi}[\hat{j}]$. Using the full linearity of FLP, we can the honest aggregator z's verifier share vfs in terms of $\mathit{jr}, \mathit{qr}, \sigma$ and the corrupt aggregators' shares, since:

$$\begin{aligned} \mathsf{Query}(\mathit{inp}, \pi, \mathit{jr}; \mathit{qr}) &= \mathsf{Query}(\vec{x}[z], \vec{\pi}[z], \mathit{jr}; \mathit{qr}) && (18) \\ &\quad + \sum_{\hat{j}\in\mathcal{T}} \mathsf{Query}(\vec{x}[\hat{j}], \vec{\pi}[\hat{j}], \mathit{jr}; \mathit{qr}) && (19) \end{aligned}$$

This revision to the game does not change the outcome of the experiment. However, since we do not have access to the prover randomness generated by $\mathsf{View}_{\mathsf{FLP}}(\mathit{inp})$, we can no longer consistently program the random oracle (see 19). Fortunately, to trigger this inconsistency, the adversary would have to guess the seed ps used to to program it prior to calling $\underline{\mathsf{Shard}}$. It follows that

$$\Pr[\underline{\mathsf{G}}_7(B)] \le \Pr[\underline{\mathsf{G}}_8(B)] + \frac{q_4 q_{\mathsf{Shard}}}{2^{\kappa}}. \tag{20}$$

Let S be the simulator hypothesized by δ-privacy of FLP. In the right panel of Figure 23 we define a series of hybrid games that replace $\mathsf{View}_{\mathsf{FLP}}$ with a simulator S for the privacy of FLP. Recall from Section 2 that S outputs a string $\mathit{jr} \,\|\, \mathit{qr} \,\|\, \sigma$. In $\underline{\mathsf{G}}_9^i(B)$, the first $i-1$ queries to $\underline{\mathsf{Shard}}$ generate $\mathit{jr}, \mathit{qr}, \sigma$ by calling $S()$; the remaining queries call $\mathsf{View}_{\mathsf{FLP}}$ instead. This means that $\underline{\mathsf{G}}_9^1$ is identical to $\underline{\mathsf{G}}_8$, so

$$\Pr\left[\,\underline{\mathsf{G}}_8(B)\,\right] = \Pr\left[\,\underline{\mathsf{G}}_9^1(B)\,\right]. \tag{21}$$

For every $v \in \mathbb{F}^{jl\times ql\times v}$, we let $p_{i,v}$ denote the probability that B wins hybrid $\underline{\mathsf{G}}_9^i$, conditioned on the event that the i^{th} query to $\underline{\mathsf{Shard}}$ sets $v = \mathit{jr} \,\|\, \mathit{qr} \,\|\, \sigma$. A union bound over all v shows that

$$\Pr[\underline{\mathsf{G}}_9^i(B)] = \sum_{v\in\mathbb{F}^{jl\times ql\times v}} p_{i,v}. \tag{22}$$

```
$\underline{\mathsf{Shard}}(\hat{k} \in \mathbb{N}, m_0, m_1 \in \mathcal{I})$:        Game $\mathsf{G}_2$ $\boxed{\mathsf{G}_3}$
if Used$[\hat{k}] \neq \bot$: ret $\bot$
$n \leftarrow\$ \mathcal{N} \setminus \mathcal{N}^*$; $\mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}$
$\vec{x} \leftarrow\$ (\mathbb{F}^n)^s$; $\vec{\pi} \leftarrow\$ (\mathbb{F}^m)^s$
$(blind_1, \ldots, blind_s) \leftarrow\$ (\{0,1\}^\kappa)^s$
$(xseed_2, \ldots, xseed_s) \leftarrow\$ (\{0,1\}^\kappa)^{s-1}$
$(pseed_2, \ldots, pseed_s) \leftarrow\$ (\{0,1\}^\kappa)^{s-1}$
$\vec{rseed} \leftarrow\$ (\{0,1\}^\kappa)^s$
for $\hat{j} \in [s]$:
  $\mathsf{PO}_2(xseed_{\hat{j}}, \hat{j}, \vec{x}[\hat{j}])$
  $\mathsf{PO}_3(pseed_{\hat{j}}, \hat{j}, \vec{\pi}[\hat{j}])$
  $\mathsf{PO}_7(blind_{\hat{j}}, \hat{j} \parallel n \parallel \vec{x}[\hat{j}], \vec{rseed}[\hat{j}])$
$\mathsf{PO}_7(blind_1, 1 \parallel n \parallel \vec{x}[1], \vec{rseed}[1])$
$inp \leftarrow \mathsf{Encode}(m_b)$
for $\hat{j} \in [2..s]$:
  $blind_{\hat{j}}, xseed_{\hat{j}}, pseed_{\hat{j}} \leftarrow\$ \{0,1\}^\kappa$
  $\vec{x}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_2(xseed_{\hat{j}}, \hat{j})$
  $\vec{rseed}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_7(blind_{\hat{j}}, \hat{j} \parallel n \parallel \vec{x}[\hat{j}])$
$\vec{x}[1] \leftarrow inp - \sum_{\hat{j}=2}^{s} \vec{x}[\hat{j}]$
$blind_1 \leftarrow\$ \{0,1\}^\kappa$; $ps \leftarrow\$ \{0,1\}^\kappa$
$\vec{rseed}[1] \leftarrow \underline{\mathsf{RO}}_7(blind_1, 1 \parallel n \parallel \vec{x}[1])$
$jseed \leftarrow\$ \{0,1\}^\kappa$
$jr \leftarrow\$ \mathbb{F}^{jl}$; $pr \leftarrow\$ \mathbb{F}^{pl}$; $qr \leftarrow\$ \mathbb{F}^{ql}$
Rand$[1, jseed, \varepsilon] \leftarrow jr$
Rand$[4, ps, \varepsilon] \leftarrow pr$
Rand$[5, sk_z, n] \leftarrow qr$
Rand$[6, 0^\kappa, \vec{rseed}] \leftarrow jseed$
$\vec{\pi}[1] \leftarrow \mathsf{Prove}(inp, jr\,; pr)$
$\vec{\pi}[1] \leftarrow \vec{\pi}[1] - \sum_{\hat{j}=2}^{s} \underline{\mathsf{RO}}_3(pseed_{\hat{j}}, \hat{j})$
$\vec{x}[1] \leftarrow (\vec{x}[1], \vec{\pi}[1], blind_1)$
for $\hat{j} \in [2..s]$:
  $\vec{x}[\hat{j}] \leftarrow (xseed_{\hat{j}}, pseed_{\hat{j}}, blind_{\hat{j}})$
Pub$[\hat{k}] \leftarrow \vec{rseed}$
In$[\hat{k}, \cdot] \leftarrow \vec{x}$
In$[\hat{k}, z] \leftarrow (\vec{x}[z], jseed, jr, qr)$
Used$[\hat{k}] \leftarrow (n, m_0, m_1)$
ret $(n, \text{Pub}[\hat{k}], (\text{In}[\hat{k}, \hat{j}])_{\hat{j} \in \mathcal{T}})$
```

```
Algorithm $\mathsf{PO}_i(seed, cntxt, out)$:
if Rand$[i, seed, cntxt] = \bot$:
  Rand$[i, seed, cntxt] \leftarrow out$
else:
  $\mathtt{bad}_4 \leftarrow \mathtt{true}$
```

```
Algorithm $\mathsf{PO}_i(seed, cntxt, out)$:        Game $\mathsf{G}_3$ $\boxed{\mathsf{G}_4}$
if Rand$[i, seed, cntxt] = \bot$:
  Rand$[i, seed, cntxt] \leftarrow out$
else:
  $\mathtt{bad}_4 \leftarrow \mathtt{true}$; $\boxed{\text{Rand}[i, seed, cntxt] \leftarrow out}$
```

```
$\underline{\mathsf{Shard}}(\hat{k} \in \mathbb{N}, m_0, m_1 \in \mathcal{I})$:        Game $\mathsf{G}_5$
if Used$[\hat{k}] \neq \bot$: ret $\bot$
$n \leftarrow\$ \mathcal{N} \setminus \mathcal{N}^*$; $\mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}$
$\vec{x} \leftarrow\$ (\mathbb{F}^n)^s$; $\vec{\pi} \leftarrow\$ (\mathbb{F}^m)^s$
$(blind_1, \ldots, blind_s) \leftarrow\$ (\{0,1\}^\kappa)^s$
$(xseed_2, \ldots, xseed_s) \leftarrow\$ (\{0,1\}^\kappa)^{s-1}$
$(pseed_2, \ldots, pseed_s) \leftarrow\$ (\{0,1\}^\kappa)^{s-1}$
$\vec{rseed} \leftarrow\$ (\{0,1\}^\kappa)^s$
for $\hat{j} \in [s]$:
  Rand$[2, xseed_{\hat{j}}, \hat{j}] \leftarrow \vec{x}[\hat{j}]$
  Rand$[3, pseed_{\hat{j}}, \hat{j}] \leftarrow \vec{\pi}[\hat{j}]$
  Rand$[7, blind_{\hat{j}}, \hat{j} \parallel n \parallel \vec{x}[\hat{j}]] \leftarrow \vec{rseed}[\hat{j}]$
Rand$[7, blind_1, 1 \parallel n \parallel \vec{x}[1]] \leftarrow \vec{rseed}[1]$
$inp \leftarrow \mathsf{Encode}(m_b)$
$\vec{x}[1] \leftarrow inp - \sum_{\hat{j}=2}^{s} \vec{x}[\hat{j}]$
$ps \leftarrow\$ \{0,1\}^\kappa$
$jseed \leftarrow\$ \{0,1\}^\kappa$
$jr \leftarrow\$ \mathbb{F}^{jl}$; $pr \leftarrow\$ \mathbb{F}^{pl}$; $qr \leftarrow\$ \mathbb{F}^{ql}$
Rand$[1, jseed, \varepsilon] \leftarrow jr$
Rand$[4, ps, \varepsilon] \leftarrow pr$
Rand$[5, sk_z, n] \leftarrow qr$
Rand$[6, 0^\kappa, \vec{rseed}] \leftarrow jseed$
$\vec{\pi}[1] \leftarrow \mathsf{Prove}(inp, jr\,; pr)$
$\vec{\pi}[1] \leftarrow \vec{\pi}[1] - \sum_{\hat{j}=2}^{s} \vec{\pi}[\hat{j}]$
$\vec{x}[1] \leftarrow (\vec{x}[1], \vec{\pi}[1], blind_1)$
for $\hat{j} \in [2..s]$:
  $\vec{x}[\hat{j}] \leftarrow (xseed_{\hat{j}}, pseed_{\hat{j}}, blind_{\hat{j}})$
Pub$[\hat{k}] \leftarrow \vec{rseed}$
In$[\hat{k}, \cdot] \leftarrow \vec{x}$
In$[\hat{k}, z] \leftarrow (\vec{x}[z], jseed, jr, qr)$
Used$[\hat{k}] \leftarrow (n, m_0, m_1)$
ret $(n, \text{Pub}[\hat{k}], (\text{In}[\hat{k}, \hat{j}])_{\hat{j} \in \mathcal{T}})$
```

Figure 21: Games $\underline{\mathsf{G}}_3$ (left), $\underline{\mathsf{G}}_4$ (top-right), and $\underline{\mathsf{G}}_5$ (bottom-right) for the proof of Theorem 2. Only the $\underline{\mathsf{Shard}}$ is shown, as this is the only object that changes in each game hop.

```
$\underline{\text{Shard}}(\hat{k} \in \mathbb{N}, m_0, m_1 \in \mathcal{I})$:        Game $\underline{\mathsf{G}}_5$ $\boxed{\underline{\mathsf{G}}_6}$
  if $\text{Used}[\hat{k}] \neq \bot$: ret $\bot$
  $n \leftarrow\$ \mathcal{N} \setminus \mathcal{N}^*$; $\mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}$
  $\vec{x} \leftarrow\$ (\mathbb{F}^n)^s$; $\vec{\pi} \leftarrow\$ (\mathbb{F}^m)^s$
  $(blind_1, \ldots, blind_s) \leftarrow\$ (\{0,1\}^\kappa)^s$
  $(xseed_2, \ldots, xseed_s) \leftarrow\$ (\{0,1\}^\kappa)^{s-1}$
  $(pseed_2, \ldots, pseed_s) \leftarrow\$ (\{0,1\}^\kappa)^{s-1}$
  $\vec{rseed} \leftarrow\$ (\{0,1\}^\kappa)^s$
  $jr \leftarrow\$ \mathbb{F}^{jl}$; $pr \leftarrow\$ \mathbb{F}^{pl}$
  $inp \leftarrow \mathsf{Encode}(m_b)$
  $\pi \leftarrow \mathsf{Prove}(inp, jr\,; pr)$
  $\vec{x}[z] \leftarrow inp - \sum_{\hat{j} \in \mathcal{T}} \vec{x}[\hat{j}]$
  $\vec{\pi}[z] \leftarrow \pi - \sum_{\hat{j} \in \mathcal{T}} \vec{\pi}[\hat{j}]$
  for $\hat{j} \in [s]$:
    $\text{Rand}[2, xseed_{\hat{j}}, \hat{j}] \leftarrow \vec{x}[\hat{j}]$
    $\text{Rand}[3, pseed_{\hat{j}}, \hat{j}] \leftarrow \vec{\pi}[\hat{j}]$
    $\text{Rand}[7, blind_{\hat{j}}, \hat{j} \parallel n \parallel \vec{x}[\hat{j}]] \leftarrow \vec{rseed}[\hat{j}]$
  $\text{Rand}[7, blind_1, 1 \parallel n \parallel \vec{x}[1]] \leftarrow \vec{rseed}[1]$
  $inp \leftarrow \mathsf{Encode}(m_b)$
  $\vec{x}[1] \leftarrow inp - \sum_{\hat{j}=2}^{s} \vec{x}[\hat{j}]$
  $ps \leftarrow\$ \{0,1\}^\kappa$
  $jseed \leftarrow\$ \{0,1\}^\kappa$
  $jr \leftarrow\$ \mathbb{F}^{jl}$; $pr \leftarrow\$ \mathbb{F}^{pl}$;  $qr \leftarrow\$ \mathbb{F}^{ql}$
  $\text{Rand}[1, jseed, \varepsilon] \leftarrow jr$
  $\text{Rand}[4, ps, \varepsilon] \leftarrow pr$
  $\text{Rand}[5, sk_z, n] \leftarrow qr$
  $\text{Rand}[6, 0^\kappa, \vec{rseed}] \leftarrow jseed$
  $\vec{\pi}[1] \leftarrow \mathsf{Prove}(inp, jr\,; pr)$
  $\vec{\pi}[1] \leftarrow \vec{\pi}[1] - \sum_{\hat{j}=2}^{s} \vec{\pi}[\hat{j}]$
  $\vec{x}[1] \leftarrow (\vec{x}[1], \vec{\pi}[1], blind_1)$
  for $\hat{j} \in [2..s]$:
    $\vec{x}[\hat{j}] \leftarrow (xseed_{\hat{j}}, pseed_{\hat{j}}, blind_{\hat{j}})$
  $\text{Pub}[\hat{k}] \leftarrow \vec{rseed}$
  $\text{In}[\hat{k}, \cdot] \leftarrow \vec{x}$
  $\text{In}[\hat{k}, z] \leftarrow (\vec{x}[z], jseed, jr, qr)$
  $\text{Used}[\hat{k}] \leftarrow (n, m_0, m_1)$
  ret $(n, \text{Pub}[\hat{k}], (\text{In}[\hat{k}, \hat{j}])_{\hat{j} \in \mathcal{T}})$
```

```
$\underline{\text{Shard}}(\hat{k} \in \mathbb{N}, m_0, m_1 \in \mathcal{I})$:        Game $\underline{\mathsf{G}}_6$ $\boxed{\underline{\mathsf{G}}_7}$
  if $\text{Used}[\hat{k}] \neq \bot$: ret $\bot$
  $n \leftarrow\$ \mathcal{N} \setminus \mathcal{N}^*$; $\mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}$
  $\vec{x} \leftarrow\$ (\mathbb{F}^n)^s$; $\vec{\pi} \leftarrow\$ (\mathbb{F}^m)^s$
  $(blind_1, \ldots, blind_s) \leftarrow\$ (\{0,1\}^\kappa)^s$
  $(xseed_2, \ldots, xseed_s) \leftarrow\$ (\{0,1\}^\kappa)^{s-1}$
  $(pseed_2, \ldots, pseed_s) \leftarrow\$ (\{0,1\}^\kappa)^{s-1}$
  $\vec{rseed} \leftarrow\$ (\{0,1\}^\kappa)^s$; $jr \leftarrow\$ \mathbb{F}^{jl}$; $pr \leftarrow\$ \mathbb{F}^{pl}$
  $inp \leftarrow \mathsf{Encode}(m_b)$
  $\pi \leftarrow \mathsf{Prove}(inp, jr\,; pr)$
  $\vec{x}[z] \leftarrow inp - \sum_{\hat{j} \in \mathcal{T}} \vec{x}[\hat{j}]$
  $\vec{\pi}[z] \leftarrow \pi - \sum_{\hat{j} \in \mathcal{T}} \vec{\pi}[\hat{j}]$
  for $\hat{j} \in [s]$:
    $\text{Rand}[2, xseed_{\hat{j}}, \hat{j}] \leftarrow \vec{x}[\hat{j}]$
    $\text{Rand}[3, pseed_{\hat{j}}, \hat{j}] \leftarrow \vec{\pi}[\hat{j}]$
    $\text{Rand}[7, blind_{\hat{j}}, \hat{j} \parallel n \parallel \vec{x}[\hat{j}]] \leftarrow \vec{rseed}[\hat{j}]$
  $\text{Rand}[7, blind_1, 1 \parallel n \parallel \vec{x}[1]] \leftarrow \vec{rseed}[1]$
  $ps \leftarrow\$ \{0,1\}^\kappa$; $jseed \leftarrow\$ \{0,1\}^\kappa$; $qr \leftarrow\$ \mathbb{F}^{ql}$
  $\text{Rand}[1, jseed, \varepsilon] \leftarrow jr$; $\text{Rand}[4, ps, \varepsilon] \leftarrow pr$
  $\text{Rand}[5, sk_z, n] \leftarrow qr$; $\text{Rand}[6, 0^\kappa, \vec{rseed}] \leftarrow jseed$
  $\vec{x}[1] \leftarrow (\vec{x}[1], \vec{\pi}[1], blind_1)$
  for $\hat{j} \in [2..s]$: $\vec{x}[\hat{j}] \leftarrow (xseed_{\hat{j}}, pseed_{\hat{j}}, blind_{\hat{j}})$
  $\text{Pub}[\hat{k}] \leftarrow \vec{rseed}$; $\text{In}[\hat{k}, \cdot] \leftarrow \vec{x}$
  $\text{In}[\hat{k}, z] \leftarrow (\vec{x}[z], jseed, jr, qr)$
  $vfs \leftarrow \mathsf{Query}(\vec{x}[z], \vec{\pi}[z], jr; qr)$
  $\text{In}[\hat{k}, \hat{j}] \leftarrow (vfs, \vec{rseed}[z], \mathsf{Truncate}(\vec{x}[z]), jseed)$
  $\text{Used}[\hat{k}] \leftarrow (n, m_0, m_1)$
  ret $(n, \text{Pub}[\hat{k}], (\text{In}[\hat{k}, \hat{j}])_{\hat{j} \in \mathcal{T}})$

$\underline{\text{Prep}}(\hat{i} \in \mathbb{N}, \hat{j} \in \{z\}, \hat{k} \in \mathbb{N}, \vec{msg} \in \mathcal{M}^*)$:
  if $\text{Status}[\hat{i}, \hat{j}] \neq$ running or $\text{In}[\hat{k}, \hat{j}] = \bot$: ret $\bot$
  if $\text{St}[\hat{i}, \hat{j}, \hat{k}] = \bot$:
    $\text{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow \text{Setup}[\hat{i}, \hat{j}]$; $\vec{msg} \leftarrow (\text{Pub}[\hat{k}], )$
  $(n, m_0, m_1) \leftarrow \text{Used}[\hat{k}]$
  if $\text{St}[\hat{i}, \hat{j}, \hat{k}] = \varepsilon$: // Process initial message from client
    $(x, jseed, jr, qr) \leftarrow \text{In}[\hat{k}, \hat{j}]$
    $(inp, \pi, blind) \leftarrow \mathsf{Unpack}(\hat{j}, x)$; $(\vec{rseed}, ) \leftarrow \vec{msg}$
    $\vec{rseed}[\hat{j}] \leftarrow \underline{\mathsf{RO}}_7(blind, \hat{j} \parallel n \parallel inp)$
    $msg \leftarrow (\mathsf{Query}(inp, \pi, jr; qr), \vec{rseed}[\hat{j}])$
    $\text{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow (jseed, \mathsf{Truncate}(inp))$
    $(vfs, rseed, y, jseed) \leftarrow \text{In}[\hat{k}, \hat{j}]$
    $msg \leftarrow (vfs, rseed)$; $\text{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow (jseed, y)$
    ret (running, $msg$)
  // Process broadcast messages from aggregators
  $(jseed, y) \leftarrow \text{St}[\hat{i}, \hat{j}, \hat{k}]$; $(\vec{vfs}[\hat{j}], \vec{rseed}[\hat{j}])_{\hat{j} \in [s]} \leftarrow \vec{msg}$
  $acc \leftarrow \mathsf{Decide}(\sum_{\hat{j}=1}^{s} \vec{vfs}[\hat{j}])$; $\text{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow \bot$
  if $acc = 0$ or $jseed \neq \underline{\mathsf{RO}}_6(0^\kappa, \vec{rseed})$: ret (failed, $\bot$)
  $\text{Out}[\hat{i}, \hat{j}, \hat{k}] \leftarrow y$
  $\text{Batch}_0[\hat{i}, \hat{j}, \hat{k}] \leftarrow m_0$; $\text{Batch}_1[\hat{i}, \hat{j}, \hat{k}] \leftarrow m_1$
  ret (finished, $\bot$)
```

Figure 22: Game $\underline{\mathsf{G}}_6$ (left) and game $\underline{\mathsf{G}}_7$ (right) for the proof of Theorem 2.

```
Shard($\hat{k} \in \mathbb{N}, m_0, m_1 \in \mathcal{I}$):        Game $\underline{\mathsf{G}}_7$ $\boxed{\underline{\mathsf{G}}_8}$
  if Used[$\hat{k}$] $\neq \bot$: ret $\bot$
  $n \leftarrow\$ \mathcal{N} \setminus \mathcal{N}^*$; $\mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}$
  $\vec{x} \leftarrow\$ (\mathbb{F}^n)^s$; $\vec{\pi} \leftarrow\$ (\mathbb{F}^m)^s$
  $(blind_1, \ldots, blind_s) \leftarrow\$ (\{0,1\}^\kappa)^s$
  $(xseed_2, \ldots, xseed_s) \leftarrow\$ (\{0,1\}^\kappa)^{s-1}$
  $(pseed_2, \ldots, pseed_s) \leftarrow\$ (\{0,1\}^\kappa)^{s-1}$
  $\vec{rseed} \leftarrow\$ (\{0,1\}^\kappa)^s$; $jr \leftarrow\$ \mathbb{F}^{jl}$; $pr \leftarrow\$ \mathbb{F}^{pl}$
  $inp \leftarrow \mathsf{Encode}(m_b)$
  $\pi \leftarrow \mathsf{Prove}(inp, jr\,; pr)$
  $\boxed{jr \,\|\, \|\, qr \,\|\, \sigma \leftarrow\$ \mathsf{View}_{\mathsf{FLP}}(inp)}$
  $\vec{x}[z] \leftarrow inp - \sum_{\hat{j} \in \mathcal{T}} \vec{x}[\hat{j}]$
  $\vec{\pi}[z] \leftarrow \pi - \sum_{\hat{j} \in \mathcal{T}} \vec{\pi}[\hat{j}]$
  for $\hat{j} \in [s]$:
    Rand$[2, xseed_{\hat{j}}, \hat{j}] \leftarrow \vec{x}[\hat{j}]$
    Rand$[3, pseed_{\hat{j}}, \hat{j}] \leftarrow \vec{\pi}[\hat{j}]$
    Rand$[7, blind_{\hat{j}}, \hat{j} \,\|\, n \,\|\, \vec{x}[\hat{j}]] \leftarrow \vec{rseed}[\hat{j}]$
  Rand$[7, blind_1, 1 \,\|\, n \,\|\, \vec{x}[1]] \leftarrow \vec{rseed}[1]$
  $ps \leftarrow\$ \{0,1\}^\kappa$; $jseed \leftarrow\$ \{0,1\}^\kappa$; $qr \leftarrow\$ \mathbb{F}^{ql}$
  Rand$[1, jseed, \varepsilon] \leftarrow jr$; Rand$[4, ps, \varepsilon] \leftarrow pr$
  Rand$[5, sk_z, n] \leftarrow qr$; Rand$[6, 0^\kappa, \vec{rseed}] \leftarrow jseed$
  $\vec{x}[1] \leftarrow (\vec{x}[1], \vec{\pi}[1], blind_1)$
  for $\hat{j} \in [2..s]$: $\vec{x}[\hat{j}] \leftarrow (xseed_{\hat{j}}, pseed_{\hat{j}}, blind_{\hat{j}})$
  Pub$[\hat{k}] \leftarrow \vec{rseed}$; In$[\hat{k}, \cdot] \leftarrow \vec{x}$
  $vfs \leftarrow \mathsf{Query}(\vec{x}[z], \vec{\pi}[z], jr; qr)$
  $\boxed{vfs \leftarrow \sigma - \sum_{\hat{j} \in \mathcal{T}} \mathsf{Query}(\vec{x}[\hat{j}], \vec{\pi}[\hat{j}], jr; qr)}$
  In$[\hat{k}, \hat{j}] \leftarrow (vfs, \vec{rseed}[z], \mathsf{Truncate}(\vec{x}[z]), jseed)$
  Used$[\hat{k}] \leftarrow (n, m_0, m_1)$
  ret $(n, \text{Pub}[\hat{k}], (\text{In}[\hat{k}, \hat{j}])_{\hat{j} \in \mathcal{T}})$
```

```
Shard($\hat{k} \in \mathbb{N}, m_0, m_1 \in \mathcal{I}$):        Game $\underline{\mathsf{G}}_8$ $\boxed{\underline{\mathsf{G}}_9^i}$
  if Used[$\hat{k}$] $\neq \bot$: ret $\bot$
  $n \leftarrow\$ \mathcal{N} \setminus \mathcal{N}^*$; $\mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}$
  $\vec{x} \leftarrow\$ (\mathbb{F}^n)^s$; $\vec{\pi} \leftarrow\$ (\mathbb{F}^m)^s$
  $(blind_1, \ldots, blind_s) \leftarrow\$ (\{0,1\}^\kappa)^s$
  $(xseed_2, \ldots, xseed_s) \leftarrow\$ (\{0,1\}^\kappa)^{s-1}$
  $(pseed_2, \ldots, pseed_s) \leftarrow\$ (\{0,1\}^\kappa)^{s-1}$
  $\vec{rseed} \leftarrow\$ (\{0,1\}^\kappa)^s$
  $inp \leftarrow \mathsf{Encode}(m_b)$
  $jr \,\|\, \|\, qr \,\|\, \sigma \leftarrow\$ \mathsf{View}_{\mathsf{FLP}}(inp)$
  $\boxed{ctr \leftarrow ctr + 1}$
  $\boxed{\text{if } ctr < i\text{: } jr \,\|\, \|\, qr \,\|\, \sigma \leftarrow\$ S(\,)}$
  $\boxed{\text{else: } jr \,\|\, \|\, qr \,\|\, \sigma \leftarrow\$ \mathsf{View}_{\mathsf{FLP}}(inp)}$
  $\vec{x}[z] \leftarrow inp - \sum_{\hat{j} \in \mathcal{T}} \vec{x}[\hat{j}]$
  for $\hat{j} \in [s]$:
    Rand$[2, xseed_{\hat{j}}, \hat{j}] \leftarrow \vec{x}[\hat{j}]$
    Rand$[3, pseed_{\hat{j}}, \hat{j}] \leftarrow \vec{\pi}[\hat{j}]$
    Rand$[7, blind_{\hat{j}}, \hat{j} \,\|\, n \,\|\, \vec{x}[\hat{j}]] \leftarrow \vec{rseed}[\hat{j}]$
  Rand$[7, blind_1, 1 \,\|\, n \,\|\, \vec{x}[1]] \leftarrow \vec{rseed}[1]$
  $jseed \leftarrow\$ \{0,1\}^\kappa$
  Rand$[1, jseed, \varepsilon] \leftarrow jr$
  Rand$[5, sk_z, n] \leftarrow qr$; Rand$[6, 0^\kappa, \vec{rseed}] \leftarrow jseed$
  $\vec{x}[1] \leftarrow (\vec{x}[1], \vec{\pi}[1], blind_1)$
  for $\hat{j} \in [2..s]$: $\vec{x}[\hat{j}] \leftarrow (xseed_{\hat{j}}, pseed_{\hat{j}}, blind_{\hat{j}})$
  Pub$[\hat{k}] \leftarrow \vec{rseed}$; In$[\hat{k}, \cdot] \leftarrow \vec{x}$
  $vfs \leftarrow \sigma -$
        $\sum_{\hat{j} \in \mathcal{T}} \mathsf{Query}(\vec{x}[\hat{j}], \vec{\pi}[\hat{j}], jr; qr)$
  In$[\hat{k}, \hat{j}] \leftarrow (vfs, \vec{rseed}[z], \mathsf{Truncate}(\vec{x}[z]), jseed)$
  Used$[\hat{k}] \leftarrow (n, m_0, m_1)$
  ret $(n, \text{Pub}[\hat{k}], (\text{In}[\hat{k}, \hat{j}])_{\hat{j} \in \mathcal{T}})$
```

Figure 23: Game $\underline{\mathsf{G}}_8$ (left) and game $\underline{\mathsf{G}}_9$ for the proof of Theorem 2.

We are now ready to bound the quantity $\Pr[\underline{\mathsf{G}}_9^{i+1}(B)] - \Pr[\underline{\mathsf{G}}_9^{i}(B)]$. The two games $\underline{\mathsf{G}}_9^{i+1}$ and $\underline{\mathsf{G}}_9^{i}$ differ only in the tuple v chosen by of the $(i+1)^{\text{th}}$ query to $\underline{\mathsf{Shard}}$: the former calls $\mathsf{View}_{\mathsf{FLP}}$ and the latter calls S. We therefore decompose both probabilities over the possible choices of v, and substitute in the statement

$$\sum_{v \in \mathbb{F}^{jl \times ql \times v}} \left| \Pr\left[\, \mathsf{View}_{\mathsf{FLP}}(inp) = v \,\right] - \Pr\left[\, S(\,) = v \,\right] \right| \leq \delta$$

that follows from the δ-privacy of FLP for all inp. Since $p_{i,v} \leq 1$ for all i and v and $\Pr[\mathsf{View}_{\mathsf{FLP}}(inp) =$

$v] - \Pr[S(\,) = v] \le |\Pr[\mathsf{View}_{\mathsf{FLP}}(\mathit{inp}) = v] - \Pr[S(\,) = v]|$:

$$\begin{aligned}
\Pr[\underline{\mathsf{G}}_9^{i+1}(B)] - \Pr[\underline{\mathsf{G}}_9^{i}(B)] = \\
&\sum_v p_{i,v} \cdot \Pr[\mathsf{View}_{\mathsf{FLP}}(\mathit{inp}) = v] - p_{i,v} \cdot \Pr[S(\,) = v] \\
&= \sum_v p_{i,v} \cdot (\Pr[\mathsf{View}_{\mathsf{FLP}}(\mathit{inp}) = v] - \Pr[S(\,) = v]) \\
&\le \sum_v |\Pr[\mathsf{View}_{\mathsf{FLP}}(\mathit{inp}) = v] - \Pr[S(\,) = v]| \\
&\le \delta \,.
\end{aligned}$$

A union bound over all $i \in [q_{\underline{\mathsf{Shard}}}]$ produces the final inequality:

$$\Pr[\underline{\mathsf{G}}_9^{q_{\mathsf{Shard}}}(B)] - \Pr[\underline{\mathsf{G}}_9^{1}(B)] \le \delta \cdot q_{\underline{\mathsf{Shard}}}. \tag{23}$$

Finally, we observe that game $\underline{\mathsf{G}}_9^{q_{\mathsf{Shard}}}$ can now be rewritten so that the outcome is independent of the challenge bit b. Hence

$$\Pr[\underline{\mathsf{G}}_9^{q_{\mathsf{Shard}}}(B)] = \frac{1}{2}\,. \tag{24}$$

Collecting bounds across all games and simplifying yields the theorem. □

C.3 Doplar Robustness (Theorem 3)

The proof is by a game-playing argument. We begin with the game $\underline{\mathsf{G}}_0$ defined in Figure 24 played by the given adversary A. This game was constructed from $\mathsf{Exp}_{\Pi}^{\mathrm{robust}}(A)$ by applying the following revisions. First, we have replaced Prep with its implementation, rolled out the loops in the $\underline{\mathsf{Prep}}$ oracle, and simplified some of the control flow. Second, we have removed the call to $\mathsf{refineFromShares}$ and set the purported refined measurement with the sum of the refined shares output by the calls to $\mathsf{VIDPF.VEval}$. (This is equivalent by refinement consistency of Π.) Third, we use the fact that the allowed-state $\mathsf{validSt}$ algorithm for Π only permits $\underline{\mathsf{Prep}}$ queries with unique (n, ℓ) pairs to make the contents of table Used more explicit. Finally, we lazy-evaluate each random oracle, denoted $\underline{\mathsf{RO}}_i$, with a table Rand. We use $\underline{\mathsf{RO}'}$ to denote the random oracle for VIDPF. By construction we have that

$$\mathsf{Adv}_{\Pi}^{\mathrm{robust}}(A) = \Pr\left[\,\underline{\mathsf{G}}_0(A)\,\right]. \tag{25}$$

In the remainder, we let q_i denote the number queries A makes to $\underline{\mathsf{RO}}_i$ and q_i denote the number of queries A makes to $\underline{\mathsf{RO}'}$; note that $q_{\mathsf{RG}} = q_1 + \cdots + q_6 + q'$.

Similar to the proof of Theorem 1, note that we have dropped the winning condition on line 16 of the robustness game (Figure 3). The refined measurement computed from the input shares is equal to $\Pi.\mathsf{Unshard}(1, (\Pi.\mathsf{Agg}(\vec{y_1}), \Pi.\mathsf{Agg}(\vec{y_2}))) = \vec{y_1} + \vec{y_2}$, so this condition is never met by definition.

Next, in game $\underline{\mathsf{G}}_1$ (left panel of Figure 24) we revise the definition of the $\underline{\mathsf{RO}}$ oracle so that for each $i \in \{5, 6\}$, the values of $\mathrm{Rand}[i, \mathit{seed}, \mathit{cntxt}]$ are sampled without replacement. The new game is identical to $\underline{\mathsf{G}}_0$ up to a collision in the output for either $\mathrm{Rand}[5, \cdot, \cdot]$ or $\mathrm{Rand}[6, \cdot, \cdot]$. Applying a birthday bound over all queries by A or by the $\underline{\mathsf{Prep}}$ oracle yields

$$\Pr\left[\,\underline{\mathsf{G}}_0(A)\,\right] \le \Pr\left[\,\underline{\mathsf{G}}_1(A)\,\right] + \frac{(q_5 + 2q_{\mathsf{Prep}})^2}{2^{\kappa+1}} + \frac{(q_6 + 3q_{\mathsf{Prep}})^2}{2^{\kappa+1}}\,. \tag{26}$$

Next, in game $\underline{\mathsf{G}}_2$ (right panel of Figure 25) we simplify the $\underline{\mathsf{Prep}}$ oracle by substituting aggregator $\hat{j}$'s local computation of the joint randomness seed $\mathit{jseed}_{\hat{j}}$ with a direct computation of the seed jseed from the parts ρ_1, ρ_2 computed on lines 9–10. Accordingly, We simplify the joint local randomness checks (lines 26–27) to just check if the purported hint $\overrightarrow{\mathit{rseed}}[\hat{j}]$ matches the computed part $\rho_{\hat{j}}$ (28–29). This change is only detectable to the adversary if it can find a joint randomness seed and hints such that the check succeeds, but the aggregators compute distinct $\mathit{jseed}_1 \ne \mathit{jseed}_2$. This is impossible by construction (transition from $\underline{\mathsf{G}}_1$ to $\underline{\mathsf{G}}_2$), so

```
Game $\underline{\mathsf{G}}_0(A)$ $\boxed{\underline{\mathsf{G}}_1(A)}$ :
  $sk \leftarrow\$ \mathcal{SK}$; $w \leftarrow$ false; $A^{\underline{\mathsf{RO}},\underline{\mathsf{Prep}}}()$; ret $w$

$\underline{\mathsf{Prep}}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$:
  $(\ell, \vec{pfx}) \leftarrow st$; $u \leftarrow |\vec{pfx}|$
  if $\mathrm{Used}[n, \ell] \neq \bot$: ret $\bot$
  $\mathrm{Used}[n, \ell] \leftarrow \top$
  $(pub, \vec{rseed}) \leftarrow msg_{\mathrm{Init}}$
  $(key_1, seed_1, \pi_1) \leftarrow \mathsf{Unpack}(1, \vec{x}[1], n, \ell)$
  $(key_2, seed_2, \pi_2) \leftarrow \mathsf{Unpack}(2, \vec{x}[2], n, \ell)$
  $\Delta_1 \leftarrow \underline{\mathsf{RO}}_2(seed_1, n \,\|\, \ell \,\|\, 1)$
  $\Delta_2 \leftarrow \underline{\mathsf{RO}}_2(seed_2, n \,\|\, \ell \,\|\, 2)$
  $\rho_1 \leftarrow \underline{\mathsf{RO}}_5(seed_1, n \,\|\, 1 \,\|\, pub \,\|\, key_1)$
  $\rho_2 \leftarrow \underline{\mathsf{RO}}_5(seed_2, n \,\|\, 2 \,\|\, pub \,\|\, key_2)$
  $jseed_1 \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \rho_1 \,\|\, \vec{rseed}[2])$
  $jseed_2 \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \vec{rseed}[1] \,\|\, \rho_2)$
  $jr_1 \leftarrow \underline{\mathsf{RO}}_1(jseed_1, n \,\|\, \ell)$
  $jr_2 \leftarrow \underline{\mathsf{RO}}_1(jseed_2, n \,\|\, \ell)$
  $qr \leftarrow \underline{\mathsf{RO}}_4(sk, n \,\|\, \ell \,\|\, )$
  $(h_1, \vec{y}_1) \leftarrow \mathsf{VIDPF.VEval}^{\underline{\mathsf{RO}'}}(1, pub, key_1, \vec{pfx})$
  $(h_2, \vec{y}_2) \leftarrow \mathsf{VIDPF.VEval}^{\underline{\mathsf{RO}'}}(2, pub, key_2, \vec{pfx})$
  $\vec{y} \leftarrow \vec{y}_1 + \vec{y}_2$
  $inp_1 \leftarrow \sum_{i \in [u]} \vec{y}_1[i]$
  $inp_2 \leftarrow \sum_{i \in [u]} \vec{y}_2[i]$
  $\sigma_1 \leftarrow \mathsf{DFLP.Query}(inp_1, \Delta_1, \pi_1, jr_1;\ qr)$
  $\sigma_2 \leftarrow \mathsf{DFLP.Query}(inp_2, \Delta_2, \pi_2, jr_2;\ qr)$
  $\overline{jseed} \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \rho_1 \,\|\, \rho_2)$
  $b_1 \leftarrow jseed_1 = \overline{jseed}$
  $b_2 \leftarrow jseed_2 = \overline{jseed}$
  $v \leftarrow \mathsf{VIDPF.Verify}^{\underline{\mathsf{RO}'}}(h_1, h_2)$
  $d \leftarrow \mathsf{DFLP.Decide}(\sigma_1 + \sigma_2)$
  if $\vec{y} \notin \mathcal{V}_{st_{\mathrm{Init}}}$
      and $(b_1 \wedge v \wedge d)$ or $(b_2 \wedge v \wedge d)$: $w \leftarrow$ true
  ret $(w, (msg_{\mathrm{Init}}, ((\sigma_1, \rho_1, h_1), (\sigma_2, \rho_2, h_2))))$

$\underline{\mathsf{RO}_i}(seed, cntxt)$:
  $l \leftarrow (jl, el, m, ql)$
  if $\mathrm{Rand}[i, seed, cntxt] = \bot$:
    if $i \leq 4$: $\mathrm{Rand}[i, seed, cntxt] \leftarrow\$ \mathbb{F}^{l[i]}$
    else: $\mathrm{Rand}[i, seed, cntxt] \leftarrow\$ \{0,1\}^\kappa$
      $\boxed{out \leftarrow\$ \{0,1\}^\kappa \setminus \mathcal{Q}_i;\ \mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup \{out\}}$
      $\boxed{\mathrm{Rand}[i, seed, cntxt] \leftarrow out}$
  ret $\mathrm{Rand}[i, seed, cntxt]$

$\underline{\mathsf{RO}'}(inp)$:
  if $\mathrm{Rand}'[inp] = \bot$: $\mathrm{Rand}'[inp] \leftarrow\$ \mathcal{Y}$
  ret $\mathrm{Rand}'[inp]$
```

```
$\underline{\mathsf{Prep}}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$:   $\underline{\mathsf{G}}_1$ $\boxed{\underline{\mathsf{G}}_2}$
  $(\ell, \vec{pfx}) \leftarrow st$; $u \leftarrow |\vec{pfx}|$
  if $\mathrm{Used}[n, \ell] \neq \bot$: ret $\bot$
  $\mathrm{Used}[n, \ell] \leftarrow \top$
  $(pub, \vec{rseed}) \leftarrow msg_{\mathrm{Init}}$
  $(key_1, seed_1, \pi_1) \leftarrow \mathsf{Unpack}(1, \vec{x}[1], n, \ell)$
  $(key_2, seed_2, \pi_2) \leftarrow \mathsf{Unpack}(2, \vec{x}[2], n, \ell)$
  $\Delta_1 \leftarrow \underline{\mathsf{RO}}_2(seed_1, n \,\|\, \ell \,\|\, 1)$
  $\Delta_2 \leftarrow \underline{\mathsf{RO}}_2(seed_2, n \,\|\, \ell \,\|\, 2)$
  $\rho_1 \leftarrow \underline{\mathsf{RO}}_5(seed_1, n \,\|\, 1 \,\|\, pub \,\|\, key_1)$
  $\rho_2 \leftarrow \underline{\mathsf{RO}}_5(seed_2, n \,\|\, 2 \,\|\, pub \,\|\, key_2)$
  $jseed_1 \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \rho_1 \,\|\, \vec{rseed}[2])$
  $jseed_2 \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \vec{rseed}[1] \,\|\, \rho_2)$
  $jr_1 \leftarrow \underline{\mathsf{RO}}_1(jseed_1, n \,\|\, \ell)$
  $jr_2 \leftarrow \underline{\mathsf{RO}}_1(jseed_2, n \,\|\, \ell)$
  $\boxed{jseed \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \rho_1 \,\|\, \rho_2)}$
  $\boxed{jr \leftarrow \underline{\mathsf{RO}}_1(jseed, n \,\|\, \ell)}$
  $qr \leftarrow \underline{\mathsf{RO}}_4(sk, n \,\|\, \ell \,\|\, )$
  $(h_1, \vec{y}_1) \leftarrow \mathsf{VIDPF.VEval}^{\underline{\mathsf{RO}'}}(1, pub, key_1, \vec{pfx})$
  $(h_2, \vec{y}_2) \leftarrow \mathsf{VIDPF.VEval}^{\underline{\mathsf{RO}'}}(2, pub, key_2, \vec{pfx})$
  $\vec{y} \leftarrow \vec{y}_1 + \vec{y}_2$
  $inp_1 \leftarrow \sum_{i \in [u]} \vec{y}_1[i]$
  $inp_2 \leftarrow \sum_{i \in [u]} \vec{y}_2[i]$
  $\sigma_1 \leftarrow \mathsf{DFLP.Query}(inp_1, \Delta_1, \pi_1,\ jr_1\ \boxed{jr};\ qr)$
  $\sigma_2 \leftarrow \mathsf{DFLP.Query}(inp_2, \Delta_2, \pi_2,\ jr_2\ \boxed{jr};\ qr)$
  $\overline{jseed} \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \rho_1 \,\|\, \rho_2)$
  $b_1 \leftarrow jseed_1 = \overline{jseed}$
  $b_2 \leftarrow jseed_2 = \overline{jseed}$
  $\boxed{b_1 \leftarrow \rho_1 \neq \vec{rseed}[1]}$
  $\boxed{b_2 \leftarrow \rho_2 \neq \vec{rseed}[2]}$
  $v \leftarrow \mathsf{VIDPF.Verify}^{\underline{\mathsf{RO}'}}(h_1, h_2)$
  $d \leftarrow \mathsf{DFLP.Decide}(\sigma_1 + \sigma_2)$
  if $\vec{y} \notin \mathcal{V}_{st_{\mathrm{Init}}}$
      and $(b_1 \wedge v \wedge d)$ or $(b_2 \wedge v \wedge d)$: $w \leftarrow$ true
  ret $(w, (msg_{\mathrm{Init}}, ((\sigma_1, \rho_1, h_1), (\sigma_2, \rho_2, h_2))))$
```

Figure 24: Games $\underline{\mathsf{G}}_0$, $\underline{\mathsf{G}}_1$, and $\underline{\mathsf{G}}_2$ for the proof of Theorem 3. Let $\mathcal{Y}$ denote the co-domain of the random oracle used by VIDPF.

$$\Pr\left[\,\underline{\mathsf{G}}_1(A)\,\right] = \Pr\left[\,\underline{\mathsf{G}}_2(A)\,\right] . \tag{27}$$

Next, in game $\underline{\mathsf{G}}_3$ (Figure 25), we make the following changes. First, we modify oracle $\underline{\mathsf{RO}}_4$ so that, for any query that coincides with the secret verification key sk sampled at the beginning of the game, the oracle immediately returns $\perp$ without programming the RO table. Second, we modify $\underline{\mathsf{Prep}}$ by replacing the call to

$$qr \leftarrow \underline{\mathsf{RO}}_4(sk, n \,\|\, \ell \,\|\,)$$

with

$$qr \leftarrow \mathrm{Rand}[4, sk, n \,\|\, \ell] \leftarrow\!\!\$\; \mathbb{F}^{ql} .$$

That way each call to $\underline{\mathsf{Prep}}$ samples fresh query randomness. The second change does not overwrite any value in Rand due to the first change. Thus the new game is identical to $\underline{\mathsf{G}}_2$ until the adversary makes a query to $\underline{\mathsf{RO}}_4$ with the seed equal to sk. Taking a union bound over all of A's queries, we have that

$$\Pr\left[\,\underline{\mathsf{G}}_2(A)\,\right] \le \Pr\left[\,\underline{\mathsf{G}}_3(A)\,\right] + \frac{q_4 q_{\mathsf{Prep}}}{2^{\kappa}} . \tag{28}$$

In the last game, $\underline{\mathsf{G}}_4$ (right-hand panel of Figure 25), we use the extractability of VIDPF to simplify the winning condition. First, we change how the IDPF output vector $\vec{y}$ is computed by $\underline{\mathsf{Prep}}$: If the one-hot check succeeds, i.e., v is set to 1 on line 24, then we use the extractor E to extract $(\alpha, \vec{\beta})$ from the transcript of the random oracle (7) and set $\vec{y}$ to $f_{\alpha,\vec{\beta}}(\vec{pfx})$. Second, we revise the winning condition (28) by requiring only that the sum of the elements of $\vec{y}$ is not in the delayed-input set $\mathcal{X} = \{0, 1\}$ for DFLP. In particular, we no longer require $\vec{y}$ to be one-hot for the adversary to win. (Recall that $\mathcal{V}_{st_{\mathrm{Init}}}$ is the set of one-hot vectors where the non-zero element is in $\mathcal{X}$.) These conditions are equivalent in the revised game, since (1) A cannot set w if $v = 0$, and if $v = 1$, vector $\vec{y}$ is one-hot by definition.

We claim that there exists an $O(t_A + q_{\mathsf{Prep}} t_E)$-time adversary B for which

$$\Pr\left[\,\underline{\mathsf{G}}_3(A)\,\right] \le \Pr\left[\,\underline{\mathsf{G}}_4(A)\,\right] + q_{\mathsf{Prep}} \cdot \mathsf{Adv}^{\mathrm{extract}}_{\mathsf{VIDPF},E}(B) . \tag{29}$$

The proof is by a hybrid argument. For each $i \in [q_{\mathsf{Prep}}]$ let $\underline{\mathsf{G}}'_i$ be the game $\underline{\mathsf{G}}_3$ except that only the first i queries to $\underline{\mathsf{Prep}}$ are answered in the usual way; the remaining queries are answered as they are in game $\underline{\mathsf{G}}_4$. Adversary B first samples $i \leftarrow\!\!\$\; [q_{\mathsf{Prep}}]$ then runs $\underline{\mathsf{G}}'_i(A)$ as usual, except that it simulates $\underline{\mathsf{Prep}}$ queries for one of the reports using its own game. Specifically, after unpacking IDPF public share pub and key shares key_1, key_2 on lines 4–6, it pauses the simulation, outputs (pub, key_1, key_2), and waits to be invoked again. On its second invocation, it resumes the simulation of the $\underline{\mathsf{Prep}}$ query until it reaches the computation of $\vec{y}$ on lines 23–26: At this point it queries its own $\underline{\mathsf{Eval}}$ oracle on the candidate prefixes $\vec{pfx}$ and sets $\vec{y}$ to the return value. Thereafter, it simulates the remainder of the game faithfully. if A sets $w \leftarrow \mathtt{true}$ in its game, then B guesses 1; otherwise it guesses 0.

Let δ_1^i (resp. δ_0^i) denote the probability that B samples i and guesses 1 in the VIDPF extractability experiment, conditioned on the outcome of the coin toss being 1 (resp. 0). Then for all i,

$$\mathsf{Adv}^{\mathrm{extract}}_{\mathsf{VIDPF},E}(A) \ge \frac{1}{q_{\mathsf{Prep}}} \left(\delta_1^i - \delta_0^i\right) . \tag{30}$$

Moreover, by construction we have that

$$\delta_1^i - \delta_0^i = \Pr\left[\,\underline{\mathsf{G}}'_i(A)\,\right] - \Pr\left[\,\underline{\mathsf{G}}'_{i+1}(A)\,\right] . \tag{31}$$

for all i. The claim follows from the observation that $\Pr\left[\,\underline{\mathsf{G}}_3(A)\,\right] = \Pr\left[\,\underline{\mathsf{G}}'_0(A)\,\right]$ and $\Pr\left[\,\underline{\mathsf{G}}_4(A)\,\right] = \Pr\left[\,\underline{\mathsf{G}}'_{q_{\mathsf{Prep}}}(A)\,\right]$.

Consider what A must do to set $w \leftarrow \mathtt{true}$ in game $\underline{\mathsf{G}}_4$. For some $\underline{\mathsf{Prep}}$ query, the delayed-input proof check must succeed when in fact the sum $\sum_{i \in [u]} \vec{y}[i]$ is not a valid encoded input. We bound A's advantage in game $\underline{\mathsf{G}}_4$ by a reduction to the soundness of DFLP. Recall from the definition of soundness in Section 5.2 that the malicious prover P^* first commits to an encoded input $(e, \mathit{\Delta})$, then gets a fresh joint randomness jr, then picks a proof forgery π. It wins if $\mathsf{DFLP.Decode}(e) \notin \mathcal{L}$ but the verifier deems the input valid (i.e., $\mathsf{DFLP.Decide}(\mathsf{DFLP.Query}(e, \mathit{\Delta}, \pi, jr;\ qr)) = 1$, where qr is a fresh query randomness sampled by the game).

Consider the malicious prover P^* in Figure 26. The basic idea is that P^* simulates $\underline{\mathsf{G}}_4(A)$ and extracts its commitment from queries to the random oracle. Specifically, the prover samples $i^* \leftarrow_{\$} [q_1 + q_{\mathsf{Prep}}]$ at the beginning of the game, and for the i^*-th query to $\underline{\mathsf{RO}}_1$, it attempts to compute (e, Δ) as follows (see lines 15–19).

The prover maintains a reverse look-up table for random oracle queries for computing the query randomness (i.e., $\underline{\mathsf{RO}}_4$), the joint randomness seed parts ($\underline{\mathsf{RO}}_5$), and the joint randomness seed ($\underline{\mathsf{RO}}_5$). On the i^*-th query, it looks for values n, ℓ, pub, key_1, key_2, $seed_1$, and $seed_2$ that would be used by a query to $\underline{\mathsf{Prep}}$. If successful, it uses these to construct its encoded input $(\mathsf{DFLP.Encode}(\Delta, inp^*), \Delta)$ to output in its game (20). It computes Δ as the sum of the $\Delta_{\hat{j}}$'s corresponding to that query (16–17). So how does it compute inp^*? Well, in $\underline{\mathsf{G}}_4$, the $\underline{\mathsf{Prep}}$ query corresponding to i^* evaluates IDPF keys shares at a set of candidate prefixes $\vec{pfx}$ chosen by the adversary. But because $\vec{pfx}$ is not known at this point, the best it can do is guess. It therefore chooses inp^* by sampling uniform randomly from the set $\mathcal{X} = \{0, 1\}$ of delayed-input values.

If extraction of the commitment is successful, then the prover outputs it, awaits the response from its game, and programs the table with the response jr (21). Thereafter, prover P^* runs $\underline{\mathsf{G}}_5(A)$ as usual until a $\underline{\mathsf{Prep}}$ query is made for the session (n^*, ℓ^*) that coincides with the distinguished $\underline{\mathsf{RO}}_1$ query i^*. At this point, the prover cannot compute the decision bit d and the verifier shares σ_1, σ_2 consistently, as it does not have access to the query randomness sampled by its game. Instead, it simply halts and outputs $\pi_1 + \pi_2$ as its proof forgery (35–37).

Observe that P^*'s simulation of $\underline{\mathsf{G}}_5(A)$ is perfect up until the point it it halts and outputs its forgery. This is due to the full linearity of DFLP, which allows us to substitute the computation of the query-generation algorithm secret-shared data in $\underline{\mathsf{G}}_5$ with the computation of the query-generation algorithm on plaintext inputs in the prover's soundness game. It follows that P^* wins precisely when A sets $w \leftarrow \mathtt{true}$ in the call to $\underline{\mathsf{Prep}}$ that coincides with the distinguished session. Conditioning on the probability that P^* guesses the correct call to $\underline{\mathsf{RO}}_1$, and that we guessed the value of inp^* correctly, we conclude that

$$\Pr\left[\,\underline{\mathsf{G}}_4(A)\,\right] \le 2(q_1 + q_{\mathsf{Prep}}) \cdot \epsilon\,. \tag{32}$$

The bound follows from gathering up each of the equations in simplifying.

C.4 Doplar Privacy (Theorem 4)

We begin with a game $\underline{\mathsf{G}}_0$ (Figure 27) in which we instantiate $\mathsf{Exp}^{\mathrm{priv}}_{\Pi}(A)$ in the random oracle model, in-line the sub-routines of Π, and simplify the code. Calls to RG have been replaced with a random oracle $\underline{\mathsf{RO}}$; as usual, $\underline{\mathsf{RO}}$ is implemented by lazy-evaluating a table Rand. In the remainder, we let q_i denote the number of queries A makes to $\underline{\mathsf{RO}}_i$. Another simplifying change we have made is to hard-code the index of the corrupt aggregator, which we denote by $\tilde{z}$. (We denote the honest aggregator by z.) Accordingly, we have removed the share index $\hat{j}$ from the oracle parameters and tables, as there is only one valid choice for these. (This is without loss of generality.) None of these changes impact the outcome of the experiment, so

$$\Pr\left[\,\mathsf{Exp}^{\mathrm{priv}}_{\Pi}(A)\,\right] = \Pr\left[\,\underline{\mathsf{G}}_0(A)\,\right]. \tag{33}$$

In game $\underline{\mathsf{G}}_1$ (Figure 27) we revise the $\underline{\mathsf{Shard}}$ oracle by sampling the nonce without replacement (line 5). This ensures each report has a unique nonce, which will be useful in subsequent steps. By a birthday bound, we have that

$$\Pr\left[\,\underline{\mathsf{G}}_0(A)\,\right] \le \Pr\left[\,\underline{\mathsf{G}}_1(A)\,\right] + \frac{{q_{\mathsf{Shard}}}^2}{|\mathcal{N}|}\,. \tag{34}$$

In our next step, $\underline{\mathsf{G}}_2$ (Figure 28), we modify the $\underline{\mathsf{Shard}}$ oracle such that, instead of querying the random oracle $\underline{\mathsf{RO}}$, it *programs* the random oracle using a new sub-routine, PO (31–34). This ensures that the output of $\underline{\mathsf{Shard}}$ is not correlated with the game's current state, allowing us to treat the sampled values as fresh. This has a cost, however, since if any of the values programmed by the oracle overwrite existing values in table Rand, then the adversary will end up with an inconsistent view. We can bound this by considering the probability of any one of the following events occurring:

- Seed seed_1 or seed_2 sampled on line 4 coincides with a query to $\underline{\mathsf{RO}}_2$ made by A (see lines 6–7). We write this as Rand_2 for short in the remainder.
- Seed seed_1 or seed_2 coincides with an element of Rand_5 (11–12).
- Vector $\vec{\mathit{rseed}}$ sampled on lines 11–12 coincides with an element of Rand_6 (13).
- Seed jseed sampled on line 15 coincides with an element of Rand_1 (16).
- Seed seed_2 coincides with an element of Rand_3 (18).

Because the nonces sampled by $\underline{\mathsf{Shard}}$ are unique, and because each of this oracle queries encodes the nonce, we can be certain that points programmed into the table by each $\underline{\mathsf{Shard}}$ query do not collide with one another. Indeed, it is only possible for these values to coincide with random oracle queries made by A. Apply a union bound over all q_{Shard} queries, we conclude that

$$\Pr\left[\,\underline{\mathsf{G}}_1(A)\,\right] \le \Pr\left[\,\underline{\mathsf{G}}_2(A)\,\right] + \frac{q_2 q_{\mathsf{Shard}}}{2^{\kappa-1}} + \frac{q_5 q_{\mathsf{Shard}}}{2^{\kappa-1}} + \frac{q_6 q_{\mathsf{Shard}}}{2^{2\kappa}} + \frac{q_1 q_{\mathsf{Shard}}}{2^{\kappa}}\,. \tag{35}$$

In the next step, $\underline{\mathsf{G}}_3$ (Figure 29), we substitute calls to $\mathsf{VIDPF.Gen}$ and $\mathsf{VIDPF.VEval}$ with calls to the simulator $S = (S^1_{\mathsf{VIDPF}}, S^2_{\mathsf{VIDPF}})$. The first part, S^1_{VIDPF}, is used to simulate the public share corrupt aggregator's key share (10); the second part, S^2_{VIDPF}, is used to simulate the honest aggregators one-hot check, based on the output of the first (41). After this second point, we no longer compute the honest aggregator's refined share $\vec{y}$ consistently. Instead, we compute the *corrupt aggregator's refined share* $\vec{\tilde{y}}$ and compute the the challenge input inp by subtracting the sum from the true sum for the input α_b (43–44).

There exists an adversary B for which

$$\Pr\left[\,\underline{\mathsf{G}}_2(A)\,\right] \le \Pr\left[\,\underline{\mathsf{G}}_3(A)\,\right] + q_{\mathsf{Shard}} \cdot \mathsf{Adv}^{\mathrm{priv}}_{\mathsf{VIDPF},S}(B)\,. \tag{36}$$

The proof is by a standard argument. In each hybrid game, we answer one more $\underline{\mathsf{Shard}}$ query (and the corresponding $\underline{\mathsf{Prep}}$ query) using S. Adversary B simply runs A in one of these hybrid games, chosen at random, and outputs whatever A outputs.

In game $\underline{\mathsf{G}}_4$ (Figure 30), we prepare for the $\underline{\mathsf{Shard}}$ oracle for the reduction to DFLP privacy. The primary change is that we have $\underline{\mathsf{Shard}}$ sample the query randomness qr that will be used to query the proof at each level (see line 18 in the left panel). This ensures that the query randomness is "committed" even before the query is made. We use the unpredictability of the nonce to bound the probability that this change leads to an inconsistent view of the experiment. In particular,

$$\Pr\left[\,\underline{\mathsf{G}}_3(A)\,\right] \le \Pr\left[\,\underline{\mathsf{G}}_4(A)\,\right] + \frac{\eta q_4 q_{\mathsf{Shard}}}{|\mathcal{N}|}\,. \tag{37}$$

In this step, we also make a couple of non-breaking changes. First, we in-line programming of the random oracle with the joint randomness and encoding randomness (16–17,19). Second, we store each proof and encoding randomness in tables P and D respectively. These changes are made to clarify the next step.

In game $\underline{\mathsf{G}}_5$ (Figure 30) we prepare the $\underline{\mathsf{Prep}}$ oracle by re-arranging the proof query. In particular, we run the query-generation algorithm on the plaintext encoded input and proof, and generate the verifier share that is output by subtracting from the verifier (denoted $\mathrm{V}[\hat{k}, \ell]$; see line 19 of the right panel) the verifier share generated from the corrupt aggregator's share. The adversary's view is consistent with the previous game by the full linearity of DFLP.

Lastly, in game $\underline{\mathsf{G}}_6$ (not pictured) we modify the $\underline{\mathsf{Prep}}$ oracle by replacing computation of the verifier from α_b with the DFLP-privacy simulator T. There exists an adversary C for which

$$\Pr\left[\,\underline{\mathsf{G}}_5(A)\,\right] \le \Pr\left[\,\underline{\mathsf{G}}_6(A)\,\right] + \eta q_{\mathsf{Shard}} \cdot \mathsf{Adv}^{\mathrm{priv}}_{\mathsf{DFLP},T}(C)\,. \tag{38}$$

The proof is by a hybrid argument, where each hybrid game $\underline{\mathsf{G}}'_{u,v}$ is defined as follows. For the first u reports and for the first v levels of the VIDPF tree, the verifier $\mathrm{V}[u, v]$ is generated as specified in game $\underline{\mathsf{G}}_5$ (line 19 in the right panel of Figure 30); all other verifiers are generated by T as specified in game $\underline{\mathsf{G}}_6$. By construction,

$$\Pr\left[\,\underline{\mathsf{G}}_5(A)\,\right] - \Pr\left[\,\underline{\mathsf{G}}_6(A)\,\right] = \Pr\left[\,\underline{\mathsf{G}}'_{0,0}(A)\,\right] - \Pr\left[\,\underline{\mathsf{G}}'_{q_{\mathsf{Shard}},\eta}(A)\,\right]. \tag{39}$$

Define DFLP-privacy attacker C as follows. (Refer to Figure 6.) On its first invocation, it simply outputs $\mathcal{X} = \{0, 1\}$ as the input set, as this is what is required by the game. On its next invocation, it is given joint randomness jr^* and query randomness qr^*. It proceeds by simulating A in a random hybrid game. It first samples $u^* \leftarrow\!\!\$\; [q_{\mathsf{Shard}}]$ and $v^* \leftarrow\!\!\$\; [\eta]$. It then runs $\underline{\mathsf{G}}'_{u^*,v^*}(A)$ except:

- On the u^*-th query to $\underline{\mathsf{Shard}}$, for the v^*-th level, it uses jr^* and qr^* to program the random oracles for the joint and query randomness respectively.
- When A makes a $\underline{\mathsf{Prep}}$ query corresponding to report u^* and level v^*, it halts and outputs x_b and awaits a response from its game. Upon being invoked once more on input σ, it sets $\mathrm{V}[u^*, v^*] \leftarrow \sigma$ and continues the simulation.

Finally, when A halts, C halts and returns whatever A output. Then C perfectly simulates $\underline{\mathsf{G}}'_{u^*,v^*}(A)$ when the value of its challenge bit is 1, and it perfectly simulates $\underline{\mathsf{G}}'_{u^*,v^*+1}(A)$ when its challenge bit is equal to 0. The claimed bound follows from a standard conditioning argument.

To complete the proof, we note that

$$\Pr\left[\,\underline{\mathsf{G}}_6(A)\,\right] = \frac{1}{2}\,. \tag{40}$$

Gathering up all of the terms and simplifying yields the desired bound.

```
$\underline{\mathsf{Prep}}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$:    $\underline{\mathsf{G}}_2$ $\boxed{\underline{\mathsf{G}}_3}$
$(\ell, \vec{pfx}) \leftarrow st$; $u \leftarrow |\vec{pfx}|$
if $\mathrm{Used}[n, \ell] \neq \bot$: ret $\bot$
$\mathrm{Used}[n, \ell] \leftarrow \top$
$(pub, \vec{rseed}) \leftarrow msg_{\mathrm{Init}}$
$(key_1, seed_1, \pi_1) \leftarrow \mathsf{Unpack}(1, \vec{x}[1], n, \ell)$
$(key_2, seed_2, \pi_2) \leftarrow \mathsf{Unpack}(2, \vec{x}[2], n, \ell)$
$\Delta_1 \leftarrow \underline{\mathsf{RO}}_2(seed_1, n \,\|\, \ell \,\|\, 1)$
$\Delta_2 \leftarrow \underline{\mathsf{RO}}_2(seed_2, n \,\|\, \ell \,\|\, 2)$
$\rho_1 \leftarrow \underline{\mathsf{RO}}_5(seed_1, n \,\|\, 1 \,\|\, pub \,\|\, key_1)$
$\rho_2 \leftarrow \underline{\mathsf{RO}}_5(seed_2, n \,\|\, 2 \,\|\, pub \,\|\, key_2)$
$jseed \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \rho_1 \,\|\, \rho_2)$
$jr \leftarrow \underline{\mathsf{RO}}_1(jseed, n \,\|\, \ell)$
$qr \leftarrow \underline{\mathsf{RO}}_4(sk, n \,\|\, \ell \,\|\, )$
$\boxed{qr \leftarrow \mathrm{Rand}[4, sk, n \,\|\, \ell] \leftarrow_{\$} \mathbb{F}^{ql}}$
$(h_1, \vec{y}_1) \leftarrow \mathsf{VIDPF.VEval}^{\underline{\mathsf{RO}}'}(1, pub, key_1, \vec{pfx})$
$(h_2, \vec{y}_2) \leftarrow \mathsf{VIDPF.VEval}^{\underline{\mathsf{RO}}'}(2, pub, key_2, \vec{pfx})$
$\vec{y} \leftarrow \vec{y}_1 + \vec{y}_2$
$inp_1 \leftarrow \sum_{i \in [u]} \vec{y}_1[i]$
$inp_2 \leftarrow \sum_{i \in [u]} \vec{y}_2[i]$
$\sigma_1 \leftarrow \mathsf{DFLP.Query}(inp_1, \Delta_1, \pi_1, jr;\ qr)$
$\sigma_2 \leftarrow \mathsf{DFLP.Query}(inp_2, \Delta_2, \pi_2, jr;\ qr)$
$b_1 \leftarrow \rho_1 \neq \vec{rseed}[1]$
$b_2 \leftarrow \rho_2 \neq \vec{rseed}[2]$
$v \leftarrow \mathsf{VIDPF.Verify}^{\underline{\mathsf{RO}}'}(h_1, h_2)$
$d \leftarrow \mathsf{DFLP.Decide}(\sigma_1 + \sigma_2)$
if $\vec{y} \notin \mathcal{V}_{st_{\mathrm{Init}}}$
    and $(b_1 \wedge v \wedge d)$ or $(b_2 \wedge v \wedge d)$: $w \leftarrow \mathtt{true}$
ret $(w, (msg_{\mathrm{Init}}, ((\sigma_1, \rho_1, h_1), (\sigma_2, \rho_2, h_2))))$

$\underline{\mathsf{RO}_i}(seed, cntxt)$:
$\boxed{\text{if } i = 4 \wedge seed = sk\text{: ret } \bot}$
$l \leftarrow (jl, el, m, ql)$
if $\mathrm{Rand}[i, seed, cntxt] = \bot$:
  if $i \leq 4$: $\mathrm{Rand}[i, seed, cntxt] \leftarrow_{\$} \mathbb{F}^{l[i]}$
  else:
    $out \leftarrow_{\$} \{0,1\}^\kappa \setminus \mathcal{Q}_i$; $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup \{out\}$
    $\mathrm{Rand}[i, seed, cntxt] \leftarrow out$
ret $\mathrm{Rand}[i, seed, cntxt]$
```

```
$\underline{\mathsf{Prep}}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$:    $\underline{\mathsf{G}}_3$ $\boxed{\underline{\mathsf{G}}_4}$
$(\ell, \vec{pfx}) \leftarrow st$; $u \leftarrow |\vec{pfx}|$
if $\mathrm{Used}[n, \ell] \neq \bot$: ret $\bot$
$\mathrm{Used}[n, \ell] \leftarrow \top$
$(pub, \vec{rseed}) \leftarrow msg_{\mathrm{Init}}$
$(key_1, seed_1, \pi_1) \leftarrow \mathsf{Unpack}(1, \vec{x}[1], n, \ell)$
$(key_2, seed_2, \pi_2) \leftarrow \mathsf{Unpack}(2, \vec{x}[2], n, \ell)$
$\boxed{\text{if } \mathrm{T}[n] = \bot\text{: } \mathrm{T}[n] \leftarrow_{\$} E(key_1, key_2, pub, \mathrm{Rand}')}$
$\Delta_1 \leftarrow \underline{\mathsf{RO}}_2(seed_1, n \,\|\, \ell \,\|\, 1)$
$\Delta_2 \leftarrow \underline{\mathsf{RO}}_2(seed_2, n \,\|\, \ell \,\|\, 2)$
$\rho_1 \leftarrow \underline{\mathsf{RO}}_5(seed_1, n \,\|\, 1 \,\|\, pub \,\|\, key_1)$
$\rho_2 \leftarrow \underline{\mathsf{RO}}_5(seed_2, n \,\|\, 2 \,\|\, pub \,\|\, key_2)$
$jseed \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \rho_1 \,\|\, \rho_2)$
$jr \leftarrow \underline{\mathsf{RO}}_1(jseed, n \,\|\, \ell)$
$qr \leftarrow \mathrm{Rand}[4, sk, n \,\|\, \ell] \leftarrow_{\$} \mathbb{F}^{ql}$
$(h_1, \vec{y}_1) \leftarrow \mathsf{VIDPF.VEval}^{\underline{\mathsf{RO}}'}(1, pub, key_1, \vec{pfx})$
$(h_2, \vec{y}_2) \leftarrow \mathsf{VIDPF.VEval}^{\underline{\mathsf{RO}}'}(2, pub, key_2, \vec{pfx})$
$\vec{y} \leftarrow \vec{y}_1 + \vec{y}_2$
$inp_1 \leftarrow \sum_{i \in [u]} \vec{y}_1[i]$
$inp_2 \leftarrow \sum_{i \in [u]} \vec{y}_2[i]$
$\sigma_1 \leftarrow \mathsf{DFLP.Query}(inp_1, \Delta_1, \pi_1, jr;\ qr)$
$\sigma_2 \leftarrow \mathsf{DFLP.Query}(inp_2, \Delta_2, \pi_2, jr;\ qr)$
$b_1 \leftarrow \rho_1 \neq \vec{rseed}[1]$
$b_2 \leftarrow \rho_2 \neq \vec{rseed}[2]$
$v \leftarrow \mathsf{VIDPF.Verify}^{\underline{\mathsf{RO}}'}(h_1, h_2)$
$\boxed{\text{if } v = 1\text{: } (\alpha, \vec{\beta}) \leftarrow_{\$} \mathrm{T}[n];\ \vec{y} \leftarrow f_{\alpha, \vec{\beta}}(\vec{pfx})}$
$\boxed{\text{else } \vec{y} \leftarrow \vec{y}_1 + \vec{y}_2}$
$d \leftarrow \mathsf{DFLP.Decide}(\sigma_1 + \sigma_2)$
if $\vec{y} \notin \mathcal{V}_{st_{\mathrm{Init}}}$ $\boxed{\left(\sum_{i \in [u]} \vec{y}[i]\right) \notin \mathcal{X}}$
    and $(b_1 \wedge v \wedge d)$ or $(b_2 \wedge v \wedge d)$: $w \leftarrow \mathtt{true}$
ret $(w, (msg_{\mathrm{Init}}, ((\sigma_1, \rho_1, h_1), (\sigma_2, \rho_2, h_2))))$
```

Figure 25: Games $\underline{\mathsf{G}}_3$ and $\underline{\mathsf{G}}_4$ for the proof of Theorem 3. Let $\mathcal{X} = \{0, 1\}$ denote the delayed-input set for DFLP.

```
Adversary $\mathcal{P}^*[A]()$:
  $i^* \leftarrow_{\$} [q_1 + q_{\mathsf{Prep}}]$; $n^*, \ell^* \leftarrow \bot$; $ctr \leftarrow 0$
  $sk \leftarrow_{\$} \mathcal{SK}$; $w \leftarrow \mathtt{false}$; $A^{\mathsf{ROExt},\mathsf{PrepSim}}()$

$\mathsf{ROExt}_i(seed, cntxt)$:
  if $i = 4 \wedge seed = sk$: ret $\bot$
  $l \leftarrow (jl, el, m, ql)$
  if $\mathrm{Rand}[i, seed, cntxt] = \bot$:
    if $i = 1$:
      $ctr \leftarrow ctr + 1$
      if $i = i^* \wedge$
        if $(\exists\, n, \ell, pub, key_1, key_2, \rho_1, \rho_2, seed_1, seed_2)$
            $\wedge\, \rho_1 = \mathrm{Rand}[5, seed_1, n \,\|\, 1 \,\|\, pub \,\|\, key_1]$
            $\wedge\, \rho_2 = \mathrm{Rand}[5, seed_2, n \,\|\, 2 \,\|\, pub \,\|\, key_2]$
            $\wedge\, seed = \mathrm{Rand}[6, 0^\kappa, \ell \,\|\, \rho_1, \,\|\, \rho_2]$:
          $(n^*, \ell^*) \leftarrow (n, \ell)$
          // We don't know $\vec{pfx}$, so guess what the sum will be!
          $inp^* \leftarrow_{\$} \{0,1\}$
          $\Delta_1 \leftarrow \mathsf{ROExt}_2(seed_1, n \,\|\, \ell \,\|\, 1)$
          $\Delta_2 \leftarrow \mathsf{ROExt}_2(seed_2, n \,\|\, \ell \,\|\, 2)$
          $\Delta \leftarrow \Delta_1 + \Delta_2$
          $e \leftarrow \mathsf{DFLP.Encode}(\Delta, inp^*)$
          output $(e, \Delta)$ and wait for $jr$.
          $\mathrm{Rand}[1, seed, cntxt] \leftarrow jr$
      else: $\mathrm{Rand}[1, seed, cntxt] \leftarrow_{\$} \mathbb{F}^{jl}$
    else if $i \in \{2,3,4\}$: $\mathrm{Rand}[i, seed, cntxt] \leftarrow_{\$} \mathbb{F}^{l[i]}$
    else:
      $out \leftarrow_{\$} \{0,1\}^\kappa \setminus \mathcal{Q}_i$; $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup \{out\}$
      $\mathrm{Rand}[i, seed, cntxt] \leftarrow out$
  ret $\mathrm{Rand}[i, seed, cntxt]$

$\mathsf{ROExt}'(inp)$:
  if $\mathrm{Rand}'[inp] = \bot$: $\mathrm{Rand}'[inp] \leftarrow_{\$} \mathcal{Y}$
  ret $\mathrm{Rand}'[inp]$
```

```
$\mathsf{PrepSim}(n, \vec{x}, msg_{\mathrm{Init}}, st_{\mathrm{Init}})$:
  $(\ell, \vec{pfx}) \leftarrow st$; $u \leftarrow |\vec{pfx}|$
  if $\mathrm{Used}[n, \ell] \neq \bot$: ret $\bot$
  $\mathrm{Used}[n, \ell] \leftarrow \top$
  $(pub, \vec{rseed}) \leftarrow msg_{\mathrm{Init}}$
  $(key_1, seed_1, \pi_1) \leftarrow \mathsf{Unpack}(1, \vec{x}[1], n, \ell)$
  $(key_2, seed_2, \pi_2) \leftarrow \mathsf{Unpack}(2, \vec{x}[2], n, \ell)$
  if $(n^*, \ell^*) = (n, \ell)$: output $\pi_1 + \pi_2$ and halt.
  if $\mathrm{T}[n] = \bot$: $\mathrm{T}[n] \leftarrow_{\$} E(key_1, key_2, pub, \mathrm{Rand}')$
  $\Delta_1 \leftarrow \underline{\mathsf{RO}}_2(seed_1, n \,\|\, \ell \,\|\, 1)$
  $\Delta_2 \leftarrow \underline{\mathsf{RO}}_2(seed_2, n \,\|\, \ell \,\|\, 2)$
  $\rho_1 \leftarrow \underline{\mathsf{RO}}_5(seed_1, n \,\|\, 1 \,\|\, pub \,\|\, key_1)$
  $\rho_2 \leftarrow \underline{\mathsf{RO}}_5(seed_2, n \,\|\, 2 \,\|\, pub \,\|\, key_2)$
  $jseed \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \rho_1, \,\|\, \rho_2)$
  $jr \leftarrow \underline{\mathsf{RO}}_1(jseed, n \,\|\, \ell)$
  $qr \leftarrow \mathrm{Rand}[4, sk, n \,\|\, \ell] \leftarrow_{\$} \mathbb{F}^{ql}$
  $(h_1, \vec{y}_1) \leftarrow \mathsf{VIDPF.VEval}^{\underline{\mathsf{RO}'}}(1, pub, key_1, \vec{pfx})$
  $(h_2, \vec{y}_2) \leftarrow \mathsf{VIDPF.VEval}^{\underline{\mathsf{RO}'}}(2, pub, key_2, \vec{pfx})$
  $inp_1 \leftarrow \sum_{i \in [u]} \vec{y}_1[i]$
  $inp_2 \leftarrow \sum_{i \in [u]} \vec{y}_2[i]$
  $\sigma_1 \leftarrow \mathsf{DFLP.Query}(inp_1, \Delta_1, \pi_1, jr;\ qr)$
  $\sigma_2 \leftarrow \mathsf{DFLP.Query}(inp_2, \Delta_2, \pi_2, jr;\ qr)$
  $b_1 \leftarrow \rho_1 \neq \vec{rseed}[1]$
  $b_2 \leftarrow \rho_2 \neq \vec{rseed}[2]$
  $v \leftarrow \mathsf{VIDPF.Verify}^{\underline{\mathsf{RO}'}}(h_1, h_2)$
  if $v = 1$: $(\alpha, \vec{\beta}) \leftarrow_{\$} \mathrm{T}[n]$; $\vec{y} \leftarrow f_{\alpha,\vec{\beta}}(\vec{pfx})$
  else $\vec{y} \leftarrow \vec{y}_1 + \vec{y}_2$
  $d \leftarrow \mathsf{DFLP.Decide}(\sigma_1 + \sigma_2)$
  if $\left(\sum_{i \in [u]} \vec{y}[i]\right) \notin \mathcal{X}$
     and $(b_1 \wedge v \wedge d)$ or $(b_2 \wedge v \wedge d)$: $w \leftarrow \mathtt{true}$
  ret $(w, (msg_{\mathrm{Init}}, ((\sigma_1, \rho_1, h_1), (\sigma_2, \rho_2, h_2))))$
```

Figure 26: Malicious prover P^* against the soundness of DFLP for the proof of Theorem 3.

```
Game $\underline{\mathsf{G}}_0(A)$ $\boxed{\underline{\mathsf{G}}_1(A)}$:
$(st_A, \{z\}, (sk, )) \leftarrow\$ A^{\underline{\mathsf{RO}}}(\,)$; $\tilde{z} \leftarrow 3 - z$
$b \leftarrow\$ \{0,1\}$; $b' \leftarrow\$ A^{\underline{\mathsf{RO}},\underline{\mathsf{Shard}},\underline{\mathsf{Setup}},\underline{\mathsf{Prep}},\underline{\mathsf{Agg}}}(st_A)$
ret $b = b'$

$\underline{\mathsf{Shard}}(\hat{k} \in \mathbb{N}, \alpha_0, \alpha_1 \in \mathcal{I})$:
if $\mathrm{Used}[\hat{k}] \neq \bot$: ret $\bot$
$n \leftarrow\$ \mathcal{N}$ $\boxed{n \leftarrow\$ \mathcal{N} \setminus \mathcal{N}^*; \mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}}$
// Construct the VIDPF key shares.
$seed_1, seed_2 \leftarrow\$ \{0,1\}^\kappa$
for $\ell \in [\eta]$:
  $\mathrm{D}[\hat{k}, \ell] \leftarrow \underline{\mathsf{RO}}_2(seed_1, n \,\|\, \ell \,\|\, 1)$
          $+ \underline{\mathsf{RO}}_2(seed_2, n \,\|\, \ell \,\|\, 2)$
  $\vec{\beta}[\ell] \leftarrow \mathsf{Encode}(\mathrm{D}[\hat{k}, \ell], 1)$
$(key_1, key_2, pub) \leftarrow\$ \mathsf{VIDPF.Gen}(\alpha_b, \vec{\beta})$
// Prepare the joint randomness.
$\vec{rseed}[1] \leftarrow \underline{\mathsf{RO}}_5(seed_1, n \,\|\, 1 \,\|\, pub \,\|\, key_1)$
$\vec{rseed}[2] \leftarrow \underline{\mathsf{RO}}_5(seed_2, n \,\|\, 2 \,\|\, pub \,\|\, key_2)$
// Generate the level proofs.
for $\ell \in [\eta]$:
  $jseed \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \vec{rseed})$
  $jr \leftarrow \underline{\mathsf{RO}}_1(jseed, n \,\|\, \ell)$
  $\pi \leftarrow\$ \mathsf{DFLP.Prove}(\{0,1\}, \mathrm{D}[\hat{k}, \ell], jr)$
  $\vec{pf}[\ell] \leftarrow \pi - \underline{\mathsf{RO}}_3(seed_2, n \,\|\, \ell)$
// Prepare the initial message and input shares.
$x_1 \leftarrow (key_1, seed_1, \vec{pf})$
$x_2 \leftarrow (key_2, seed_2)$
$\mathrm{In}[\hat{k}] \leftarrow x_z$
$\mathrm{Pub}[\hat{k}] \leftarrow (pub, \vec{rseed})$
$\mathrm{Used}[\hat{k}] \leftarrow (n, \alpha_0, \alpha_1)$
ret $(n, \mathrm{Pub}[\hat{k}], (x_{\tilde{z}}, ))$

$\underline{\mathsf{Setup}}(\hat{i} \in \mathbb{N}, st_{\mathrm{Init}} \in \mathcal{Q}_{\mathrm{Init}})$:
$(\ell, \vec{pfx}) \leftarrow st_{\mathrm{Init}}$
if $\mathrm{Status}[\hat{i}] \neq \bot$ or $\ell \in \mathcal{U}$ or $\vec{pfx}$ not distinct: ret $\bot$
$\mathcal{U} \leftarrow \mathcal{U} \cup \{\ell\}$
$\mathrm{Setup}[\hat{i}] \leftarrow st_{\mathrm{Init}}$; $\mathrm{Status}[\hat{i}] \leftarrow \mathtt{running}$
```

```
$\underline{\mathsf{Prep}}(\hat{i} \in \mathbb{N}, \hat{k} \in \mathbb{N}, \vec{msg} \in \mathcal{M}^*)$:
if $\mathrm{Status}[\hat{i}] \neq \mathtt{running}$ or $\mathrm{In}[\hat{k}] = \bot$: ret $\bot$
if $\mathrm{St}[\hat{i}, \hat{k}] = \bot$: $\mathrm{St}[\hat{i}, \hat{k}] \leftarrow \mathrm{Setup}[\hat{i}]$
$(n, \alpha_0, \alpha_1) \leftarrow \mathrm{Used}[\hat{k}]$
if $\mathrm{St}[\hat{i}, \hat{k}] \in \mathcal{Q}_{\mathrm{Init}}$: // Process initial message from client
  $(\ell, \vec{pfx}) \leftarrow \mathrm{St}[\hat{i}, \hat{k}]$; $u \leftarrow |\vec{pfx}|$
  $(pub, \vec{rseed}) \leftarrow \mathrm{Pub}[\hat{k}]$
  $(key, seed, \pi) \leftarrow \mathsf{Unpack}(z, \mathrm{In}[\hat{k}], n, \ell)$
  $\Delta \leftarrow \underline{\mathsf{RO}}_2(seed, n \,\|\, \ell \,\|\, z)$
  $\vec{rseed}[z] \leftarrow \underline{\mathsf{RO}}_5(seed, n \,\|\, z \,\|\, pub \,\|\, key)$
  $jseed \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \vec{rseed})$
  $jr \leftarrow \underline{\mathsf{RO}}_1(jseed, n \,\|\, \ell)$; $qr \leftarrow \underline{\mathsf{RO}}_4(sk, n \,\|\, \ell)$
  $(h, \vec{y}) \leftarrow \mathsf{VIDPF.VEval}(z, pub, key, \vec{pfx})$
  $inp \leftarrow \sum_{i \in [u]} \vec{y}[i]$
  $\sigma \leftarrow \mathsf{DFLP.Query}(inp, \Delta, \pi, jr;\ qr)$
  $msg \leftarrow (\sigma, \vec{rseed}[z], h)$
  $\mathrm{St}[\hat{i}, \hat{k}] \leftarrow (jseed, (\mathsf{DFLP.Decode}(\vec{y}[i]))_{i \in [u]})$
  ret $(\mathtt{running}, msg)$
// Process broadcast messages from aggregators
$(jseed, \vec{y}) \leftarrow \mathrm{St}[\hat{i}, \hat{k}]$; $\mathrm{St}[\hat{i}, \hat{k}] \leftarrow \bot$
$\Big((\sigma_1, rseed_1, h_1), (\sigma_2, rseed_2, h_2)\Big) \leftarrow \vec{msg}$
$acc_{\mathsf{DFLP}} \leftarrow \mathsf{DFLP.Decide}(\sigma_1 + \sigma_2)$
$acc_{\mathsf{VIDPF}} \leftarrow \mathsf{VIDPF.Verify}(h_1, h_2)$
$acc_0 \leftarrow jseed = \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, rseed_1 \,\|\, rseed_2)$
if $acc_{\mathsf{DFLP}}$ and $acc_{\mathsf{VIDPF}}$ and $acc_0$:
  $\mathrm{Out}[\hat{i}, \hat{k}] \leftarrow \vec{y}$; $\mathrm{Batch}_0[\hat{i}, \hat{k}] \leftarrow \alpha_0$; $\mathrm{Batch}_1[\hat{i}, \hat{k}] \leftarrow \alpha_1$
  ret $\mathtt{finished}$
ret $\mathtt{failed}$

$\underline{\mathsf{Agg}}(\hat{i} \in \mathbb{N})$:
if $\mathrm{Status}[\hat{i}] \neq \mathtt{running}$: ret $\bot$
$st_{\mathrm{Init}} \leftarrow \mathrm{Setup}[\hat{i}]$
if $F(st_{\mathrm{Init}}, \mathrm{Batch}_0[\hat{i}, \cdot]) \neq F(st_{\mathrm{Init}}, \mathrm{Batch}_1[\hat{i}, \cdot])$: ret $\bot$
$\mathrm{Status}[\hat{i}] \leftarrow \mathtt{finished}$
ret $\sum_{\vec{y} \in \mathrm{Out}[\hat{i}, \cdot]} \vec{y}$
```

Figure 27: Games $\underline{\mathsf{G}}_0$ and $\underline{\mathsf{G}}_1$ for the proof of Theorem 4.

```
$\underline{\mathsf{Shard}}(\hat{k} \in \mathbb{N}, \alpha_0, \alpha_1 \in \mathcal{I})$:
  if $\mathrm{Used}[\hat{k}] \neq \bot$: ret $\bot$
  $n \leftarrow\!\!\$\ \mathcal{N} \setminus \mathcal{N}^*; \mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}$
  // Construct the VIDPF key shares.
  $seed_1, seed_2 \leftarrow\!\!\$\ \{0,1\}^\kappa$
  for $\ell \in [\eta]$:
    $\mathrm{D}[\hat{k}, \ell] \leftarrow \underline{\mathsf{RO}}_2\ \boxed{\mathsf{PO}_2}(seed_1, n \,\|\, \ell \,\|\, 1)$
            $+ \underline{\mathsf{RO}}_2\ \boxed{\mathsf{PO}_2}(seed_2, n \,\|\, \ell \,\|\, 2)$
    $\vec{\beta}[\ell] \leftarrow \mathsf{Encode}(\mathrm{D}[\hat{k}, \ell], 1)$
  $(key_1, key_2, pub) \leftarrow\!\!\$\ \mathsf{VIDPF.Gen}(\alpha_b, \vec{\beta})$
  // Prepare the joint randomness.
  $\vec{rseed}[1] \leftarrow \underline{\mathsf{RO}}_5\ \boxed{\mathsf{PO}_5}(seed_1, n \,\|\, 1 \,\|\, pub \,\|\, key_1)$
  $\vec{rseed}[2] \leftarrow \underline{\mathsf{RO}}_5\ \boxed{\mathsf{PO}_5}(seed_2, n \,\|\, 2 \,\|\, pub \,\|\, key_2)$
  // Generate the level proofs.
  for $\ell \in [\eta]$:
    $jseed \leftarrow \underline{\mathsf{RO}}_6\ \boxed{\mathsf{PO}_6}(0^\kappa, \ell \,\|\, \vec{rseed})$
    $jr \leftarrow \underline{\mathsf{RO}}_1\ \boxed{\mathsf{PO}_1}(jseed, n \,\|\, \ell)$
    $\pi \leftarrow\!\!\$\ \mathsf{DFLP.Prove}(\{0,1\}, \mathrm{D}[\hat{k}, \ell], jr)$
    $\vec{pf}[\ell] \leftarrow \pi - \underline{\mathsf{RO}}_3\ \boxed{\mathsf{PO}_3}(seed_2, n \,\|\, \ell)$
  // Prepare the initial message and input shares.
  $x_1 \leftarrow (key_1, seed_1, \vec{pf})$
  $x_2 \leftarrow (key_2, seed_2)$
  $\mathrm{In}[\hat{k}] \leftarrow x_z$
  $\mathrm{Pub}[\hat{k}] \leftarrow (pub, \vec{rseed})$
  $\mathrm{Used}[\hat{k}] \leftarrow (n, \alpha_0, \alpha_1)$
  ret $(n, \mathrm{Pub}[\hat{k}], (x_{\bar{z}}, ))$

$\underline{\mathsf{RO}}_i(seed, cntxt)$:                    $\underline{\mathsf{G}}_1$ $\boxed{\underline{\mathsf{G}}_2}$
  $l \leftarrow (jl, el, m, ql)$
  if $\mathrm{Rand}[i, seed, cntxt] = \bot$:
    if $i \leq 4$: $\mathrm{Rand}[i, seed, cntxt] \leftarrow\!\!\$\ \mathbb{F}^{l[i]}$
    else: $\mathrm{Rand}[i, seed, cntxt] \leftarrow\!\!\$\ \{0,1\}^\kappa$
  ret $\mathrm{Rand}[i, seed, cntxt]$

$\mathsf{PO}_i(seed, cntxt)$:
  $l \leftarrow (jl, el, m, ql)$
  if $i \leq 4$: $\mathrm{Rand}[i, seed, cntxt] \leftarrow\!\!\$\ \mathbb{F}^{l[i]}$
  else: $\mathrm{Rand}[i, seed, cntxt] \leftarrow\!\!\$\ \{0,1\}^\kappa$
  ret $\mathrm{Rand}[i, seed, cntxt]$
```

Figure 28: Game $\underline{\mathsf{G}}_2$ for the proof of Theorem 4.

```
$\underline{\mathsf{Shard}}(\hat{k} \in \mathbb{N}, \alpha_0, \alpha_1 \in \mathcal{I})$:
  if $\mathrm{Used}[\hat{k}] \neq \bot$: ret $\bot$
  $n \leftarrow\!\!\$\ \mathcal{N} \setminus \mathcal{N}^*; \mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}$
  // Construct the VIDPF key shares.
  $\mathit{seed}_1, \mathit{seed}_2 \leftarrow\!\!\$\ \{0,1\}^\kappa$
  for $\ell \in [\eta]$:
    $\mathrm{D}[\hat{k}, \ell] \leftarrow \mathsf{PO}_2(\mathit{seed}_1, n \,\|\, \ell \,\|\, 1)$
            $+ \mathsf{PO}_2(\mathit{seed}_2, n \,\|\, \ell \,\|\, 2)$
    $\vec{\beta}[\ell] \leftarrow \mathsf{Encode}(\mathrm{D}[\hat{k}, \ell], 1)$
  $(\mathit{key}_1, \mathit{key}_2, \mathit{pub}) \leftarrow\!\!\$\ \mathsf{VIDPF.Gen}(\alpha_b, \vec{\beta})$
  $\boxed{(\mathrm{T}[\hat{k}], \mathit{pub}) \leftarrow\!\!\$\ S^1_{\mathsf{VIDPF}}(\tilde{z});\ \mathit{key}_{\tilde{z}} \leftarrow \mathrm{T}[\hat{k}];\ \mathit{key}_z \leftarrow \bot}$
  // Prepare the joint randomness.
  $\vec{\mathit{rseed}}[1] \leftarrow \mathsf{PO}_5(\mathit{seed}_1, n \,\|\, 1 \,\|\, \mathit{pub} \,\|\, \mathit{key}_1)$
  $\vec{\mathit{rseed}}[2] \leftarrow \mathsf{PO}_5(\mathit{seed}_2, n \,\|\, 2 \,\|\, \mathit{pub} \,\|\, \mathit{key}_2)$
  // Generate the level proofs.
  for $\ell \in [\eta]$:
    $\mathit{jseed} \leftarrow \mathsf{PO}_6(0^\kappa, \ell \,\|\, \vec{\mathit{rseed}})$
    $\mathit{jr} \leftarrow \mathsf{PO}_1(\mathit{jseed}, n \,\|\, \ell)$
    $\pi \leftarrow\!\!\$\ \mathsf{DFLP.Prove}(\{0,1\}, \mathrm{D}[\hat{k}, \ell], \mathit{jr})$
    $\vec{\mathit{pf}}[\ell] \leftarrow \pi - \mathsf{PO}_3(\mathit{seed}_2, n \,\|\, \ell)$
  // Prepare the initial message and input shares.
  $x_1 \leftarrow (\mathit{key}_1, \mathit{seed}_1, \vec{\mathit{pf}})$
  $x_2 \leftarrow (\mathit{key}_2, \mathit{seed}_2)$
  $\mathrm{In}[\hat{k}] \leftarrow x_z$
  $\mathrm{Pub}[\hat{k}] \leftarrow (\mathit{pub}, \vec{\mathit{rseed}})$
  $\mathrm{Used}[\hat{k}] \leftarrow (n, \alpha_0, \alpha_1)$
  ret $(n, \mathrm{Pub}[\hat{k}], (x_{\tilde{z}}, ))$
```

```
$\underline{\mathsf{Prep}}(\hat{i} \in \mathbb{N}, \hat{k} \in \mathbb{N}, \vec{\mathit{msg}} \in \mathcal{M}^*)$:        $\underline{\mathsf{G}}_2$  $\boxed{\underline{\mathsf{G}}_3}$
  if $\mathrm{Status}[\hat{i}] \neq \texttt{running}$ or $\mathrm{In}[\hat{k}] = \bot$: ret $\bot$
  if $\mathrm{St}[\hat{i}, \hat{k}] = \bot$: $\mathrm{St}[\hat{i}, \hat{k}] \leftarrow \mathrm{Setup}[\hat{i}]$
  $(n, \alpha_0, \alpha_1) \leftarrow \mathrm{Used}[\hat{k}]$
  if $\mathrm{St}[\hat{i}, \hat{k}] \in \mathcal{Q}_{\mathrm{Init}}$: // Process initial message from client
    $(\ell, \vec{\mathit{pfx}}) \leftarrow \mathrm{St}[\hat{i}, \hat{k}];\ u \leftarrow |\vec{\mathit{pfx}}|$
    $(\mathit{pub}, \vec{\mathit{rseed}}) \leftarrow \mathrm{Pub}[\hat{k}]$
    $(\mathit{key}\ \boxed{\_}, \mathit{seed}, \pi) \leftarrow \mathsf{Unpack}(z, \mathrm{In}[\hat{k}], n, \ell)$
    $\Delta \leftarrow \underline{\mathsf{RO}}_2(\mathit{seed}, n \,\|\, \ell \,\|\, z)$
    $\vec{\mathit{rseed}}[z] \leftarrow \underline{\mathsf{RO}}_5(\mathit{seed}, n \,\|\, z \,\|\, \mathit{pub} \,\|\, \mathit{key})$
    $\mathit{jseed} \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \vec{\mathit{rseed}})$
    $\mathit{jr} \leftarrow \underline{\mathsf{RO}}_1(\mathit{jseed}, n \,\|\, \ell);\ \mathit{qr} \leftarrow \underline{\mathsf{RO}}_4(\mathit{sk}, n \,\|\, \ell)$
    $(h, \vec{y}) \leftarrow \mathsf{VIDPF.VEval}(z, \mathit{pub}, \mathit{key}, \vec{\mathit{pfx}})$
    $\mathit{inp} \leftarrow \sum_{i \in [u]} \vec{y}[i]$
    $\boxed{\mathit{key}_{\tilde{z}} \leftarrow \mathrm{T}[\hat{k}]}$
    $\boxed{h \leftarrow\!\!\$\ S^2_{\mathsf{VIDPF}}(\tilde{z}, \mathit{pub}, \mathit{key}_{\tilde{z}}, \vec{\mathit{pfx}})}$
    $\boxed{(\_, \vec{\tilde{y}}) \leftarrow \mathsf{VIDPF.VEval}(\tilde{z}, \mathit{pub}, \mathit{key}_{\tilde{z}}, \vec{\mathit{pfx}})}$
    $\boxed{x_b \leftarrow |\{\vec{\mathit{pfx}}[i] : \vec{\mathit{pfx}}[i] \text{ prefixes } \alpha_b\}_{i \in [u]}|}$
    $\boxed{\mathit{inp}_b \leftarrow \mathsf{DFLP.Encode}(\Delta[\hat{k}, \ell], x_b)}$
    $\boxed{\mathit{inp} \leftarrow \mathit{inp}_b - \sum_{i \in [u]} \vec{\tilde{y}}[i]}$
    $\sigma \leftarrow \mathsf{DFLP.Query}(\mathit{inp}, \Delta, \pi, \mathit{jr};\ \mathit{qr})$
    $\mathit{msg} \leftarrow (\sigma, \vec{\mathit{rseed}}[z], h)$
    $\mathrm{St}[\hat{i}, \hat{k}] \leftarrow (\mathit{jseed}, (\mathsf{DFLP.Decode}(\vec{y}[i]))_{i \in [u]})$
    ret $(\texttt{running}, \mathit{msg})$
  // Process broadcast messages from aggregators
  $(\mathit{jseed}, \vec{y}) \leftarrow \mathrm{St}[\hat{i}, \hat{k}];\ \mathrm{St}[\hat{i}, \hat{k}] \leftarrow \bot$
  $\Big((\sigma_1, \mathit{rseed}_1, h_1), (\sigma_2, \mathit{rseed}_2, h_2)\Big) \leftarrow \vec{\mathit{msg}}$
  $\mathit{acc}_{\mathsf{DFLP}} \leftarrow \mathsf{DFLP.Decide}(\sigma_1 + \sigma_2)$
  $\mathit{acc}_{\mathsf{VIDPF}} \leftarrow \mathsf{VIDPF.Verify}(h_1, h_2)$
  $\mathit{acc}_0 \leftarrow \mathit{jseed} = \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \mathit{rseed}_1 \,\|\, \mathit{rseed}_2)$
  if $\mathit{acc}_{\mathsf{DFLP}}$ and $\mathit{acc}_{\mathsf{VIDPF}}$ and $\mathit{acc}_0$:
    $\mathrm{Out}[\hat{i}, \hat{k}] \leftarrow \vec{y};\ \mathrm{Batch}_0[\hat{i}, \hat{k}] \leftarrow \alpha_0;\ \mathrm{Batch}_1[\hat{i}, \hat{k}] \leftarrow \alpha_1$
    ret $\texttt{finished}$
  ret $\texttt{failed}$
```

Figure 29: Game $\underline{\mathsf{G}}_3$ for the proof of Theorem 4.

```
$\underline{\mathsf{Shard}}(\hat{k} \in \mathbb{N}, \alpha_0, \alpha_1 \in \mathcal{I})$:    $\underline{\mathsf{G}}_3$ $\boxed{\underline{\mathsf{G}}_4}$
if $\mathrm{Used}[\hat{k}] \neq \bot$: ret $\bot$
$n \leftarrow\!\!\$\ \mathcal{N} \setminus \mathcal{N}^*; \mathcal{N}^* \leftarrow \mathcal{N}^* \cup \{n\}$
// Construct the VIDPF key shares.
$\mathit{seed}_1, \mathit{seed}_2 \leftarrow\!\!\$\ \{0,1\}^\kappa$
 for $\ell \in [\eta]$:
   $\mathrm{D}[\hat{k}, \ell] \leftarrow \mathsf{PO}_2(\mathit{seed}_1, n \,\|\, \ell \,\|\, 1)$
           $+ \mathsf{PO}_2(\mathit{seed}_2, n \,\|\, \ell \,\|\, 2)$
$(\mathrm{T}[\hat{k}], \mathit{pub}) \leftarrow\!\!\$\ S^1_{\mathsf{VIDPF}}(\tilde{z})$; $\mathit{key}_{\tilde{z}} \leftarrow \mathrm{T}[\hat{k}]$; $\mathit{key}_z \leftarrow \bot$
// Prepare the joint randomness.
$\vec{\mathit{rseed}}[1] \leftarrow \mathsf{PO}_5(\mathit{seed}_1, n \,\|\, 1 \,\|\, \mathit{pub} \,\|\, \mathit{key}_1)$
$\vec{\mathit{rseed}}[2] \leftarrow \mathsf{PO}_5(\mathit{seed}_2, n \,\|\, 2 \,\|\, \mathit{pub} \,\|\, \mathit{key}_2)$
// Generate the level proofs.
for $\ell \in [\eta]$:
  $\mathit{jseed} \leftarrow \mathsf{PO}_6(0^\kappa, \ell \,\|\, \vec{\mathit{rseed}})$
    $\mathit{jr} \leftarrow\!\!\$\ \mathbb{F}^{jl}$; $\mathit{qr} \leftarrow\!\!\$\ \mathbb{F}^{ql}$; $\mathrm{D}[\hat{k}, \ell], \tilde{\Delta} \leftarrow\!\!\$\ \mathbb{F}^{el}$
    $\mathrm{Rand}[2, \mathit{seed}_z, n \,\|\, \ell \,\|\, z] \leftarrow \mathrm{D}[\hat{k}, \ell] - \tilde{\Delta}$
    $\mathrm{Rand}[2, \mathit{seed}_{\tilde{z}}, n \,\|\, \ell \,\|\, \tilde{z}] \leftarrow \tilde{\Delta}$
    $\mathrm{Rand}[4, \mathit{sk}, n \,\|\, \ell] \leftarrow \mathit{qr}$
    $\mathrm{Rand}[1, \mathit{jseed}, n \,\|\, \ell] \leftarrow \mathit{jr}$
    $\mathrm{P}[\hat{k}, \ell] \leftarrow\!\!\$\ \mathsf{DFLP.Prove}(\{0,1\}, \Delta, \mathit{jr})$
    $\vec{\mathit{pf}}[\ell] \leftarrow \mathrm{P}[\hat{k}, \ell] - \mathsf{PO}_3(\mathit{seed}_2, n \,\|\, \ell)$
    $\mathit{jr} \leftarrow \mathsf{PO}_1(\mathit{jseed}, n \,\|\, \ell)$
    $\pi \leftarrow\!\!\$\ \mathsf{DFLP.Prove}(\{0,1\}, \mathrm{D}[\hat{k}, \ell], \mathit{jr})$
    $\vec{\mathit{pf}}[\ell] \leftarrow \pi - \mathsf{PO}_3(\mathit{seed}_2, n \,\|\, \ell)$
// Prepare the initial message and input shares.
$x_1 \leftarrow (\mathit{key}_1, \mathit{seed}_1, \vec{\mathit{pf}})$; $x_2 \leftarrow (\mathit{key}_2, \mathit{seed}_2)$
$\mathrm{In}[\hat{k}] \leftarrow x_z$; $\mathrm{Pub}[\hat{k}] \leftarrow (\mathit{pub}, \vec{\mathit{rseed}})$
$\mathrm{Used}[\hat{k}] \leftarrow (n, \alpha_0, \alpha_1)$
ret $(n, \mathrm{Pub}[\hat{k}], (x_{\tilde{z}}, ))$
```

```
$\underline{\mathsf{Prep}}(\hat{i} \in \mathbb{N}, \hat{k} \in \mathbb{N}, \vec{\mathit{msg}} \in \mathcal{M}^*)$:    $\underline{\mathsf{G}}_4$ $\boxed{\underline{\mathsf{G}}_5}$
if $\mathrm{Status}[\hat{i}] \neq$ running or $\mathrm{In}[\hat{k}] = \bot$: ret $\bot$
if $\mathrm{St}[\hat{i}, \hat{k}] = \bot$: $\mathrm{St}[\hat{i}, \hat{k}] \leftarrow \mathrm{Setup}[\hat{i}]$
$(n, \alpha_0, \alpha_1) \leftarrow \mathrm{Used}[\hat{k}]$
if $\mathrm{St}[\hat{i}, \hat{k}] \in \mathcal{Q}_{\mathrm{Init}}$: // Process initial message from client
  $(\ell, \vec{\mathit{pfx}}) \leftarrow \mathrm{St}[\hat{i}, \hat{k}]$; $u \leftarrow |\vec{\mathit{pfx}}|$
  $(\mathit{pub}, \vec{\mathit{rseed}}) \leftarrow \mathrm{Pub}[\hat{k}]$
  $(\_, \mathit{seed}, \pi) \leftarrow \mathsf{Unpack}(z, \mathrm{In}[\hat{k}], n, \ell)$
  $\Delta \leftarrow \underline{\mathsf{RO}}_2(\mathit{seed}, n \,\|\, \ell \,\|\, z)$
  $\vec{\mathit{rseed}}[z] \leftarrow \underline{\mathsf{RO}}_5(\mathit{seed}, n \,\|\, z \,\|\, \mathit{pub} \,\|\, \mathit{key})$
  $\mathit{jseed} \leftarrow \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \vec{\mathit{rseed}})$
  $\mathit{jr} \leftarrow \underline{\mathsf{RO}}_1(\mathit{jseed}, n \,\|\, \ell)$; $\mathit{qr} \leftarrow \underline{\mathsf{RO}}_4(\mathit{sk}, n \,\|\, \ell)$
  $\mathit{key}_{\tilde{z}} \leftarrow \mathrm{T}[\hat{k}]$
  $h \leftarrow\!\!\$\ S^2_{\mathsf{VIDPF}}(\tilde{z}, \mathit{pub}, \mathit{key}_{\tilde{z}}, \vec{\mathit{pfx}})$
  $(\_, \vec{\tilde{y}}) \leftarrow \mathsf{VIDPF.VEval}(\tilde{z}, \mathit{pub}, \mathit{key}_{\tilde{z}}, \vec{\mathit{pfx}})$
  $x_b \leftarrow |\{\vec{\mathit{pfx}}[i] : \vec{\mathit{pfx}}[i] \text{ prefixes } \alpha_b\}_{i \in [u]}|$
  $\mathit{inp}_b \leftarrow \mathsf{DFLP.Encode}(\Delta[\hat{k}, \ell], x_b)$
   $\mathit{inp} \leftarrow \mathit{inp}_b - \sum_{i \in [u]} \vec{\tilde{y}}[i]$
   $\sigma \leftarrow \mathsf{DFLP.Query}(\mathit{inp}, \Delta, \pi, \mathit{jr};\ \mathit{qr})$
   $\mathrm{V}[\hat{k}, \ell] \leftarrow \mathsf{DFLP.Query}(\mathit{inp}_b, \mathrm{D}[\hat{k}, \ell], \mathrm{P}[\hat{k}, \ell], \mathit{jr};\ \mathit{qr})$
   $\sigma \leftarrow \mathrm{V}[\hat{k}, \ell] - \mathsf{DFLP.Query}(\sum_{i \in [u]} \vec{\tilde{y}}[i], \Delta, \pi, \mathit{jr};\ \mathit{qr})$
  $\mathit{msg} \leftarrow (\sigma, \vec{\mathit{rseed}}[z], h)$
  $\mathrm{St}[\hat{i}, \hat{k}] \leftarrow (\mathit{jseed}, (\mathsf{DFLP.Decode}(\vec{y}[i]))_{i \in [u]})$
  ret (running, $\mathit{msg}$)
// Process broadcast messages from aggregators
$(\mathit{jseed}, \vec{y}) \leftarrow \mathrm{St}[\hat{i}, \hat{k}]$; $\mathrm{St}[\hat{i}, \hat{k}] \leftarrow \bot$
$\Big((\sigma_1, \mathit{rseed}_1, h_1), (\sigma_2, \mathit{rseed}_2, h_2)\Big) \leftarrow \vec{\mathit{msg}}$
$\mathit{acc}_{\mathsf{DFLP}} \leftarrow \mathsf{DFLP.Decide}(\sigma_1 + \sigma_2)$
$\mathit{acc}_{\mathsf{VIDPF}} \leftarrow \mathsf{VIDPF.Verify}(h_1, h_2)$
$\mathit{acc}_0 \leftarrow \mathit{jseed} = \underline{\mathsf{RO}}_6(0^\kappa, \ell \,\|\, \mathit{rseed}_1 \,\|\, \mathit{rseed}_2)$
if $\mathit{acc}_{\mathsf{DFLP}}$ and $\mathit{acc}_{\mathsf{VIDPF}}$ and $\mathit{acc}_0$:
  $\mathrm{Out}[\hat{i}, \hat{k}] \leftarrow \vec{y}$; $\mathrm{Batch}_0[\hat{i}, \hat{k}] \leftarrow \alpha_0$; $\mathrm{Batch}_1[\hat{i}, \hat{k}] \leftarrow \alpha_1$
  ret finished
ret failed
```

Figure 30: Games $\underline{\mathsf{G}}_4$ and $\underline{\mathsf{G}}_5$ for the proof of Theorem 4.