Distributed Point Function with Constraints, Revisited*

Keyu Ji† Bingsheng Zhang† Hong-Sheng Zhou‡ Kui Ren†
†Zhejiang University. Email: {jikeyu, bingsheng, kuiren}@zju.edu.cn.
‡Virginia Commonwealth University. Email: hszhou@vcu.edu.

Abstract

Distributed Point Function (DPF) provides a way for a dealer to split a point function $f_{\alpha,\beta}$ into multiple succinctly described function-shares, where the function $f_{\alpha,\beta}$ for a special input α, returns a special output value β, and returns a fixed value 0 otherwise. As the security requirement, any strict subset of the function-shares reveals nothing about the function $f_{\alpha,\beta}$. However, each function-share can be individually evaluated on the common input x, and these evaluation results can then be merged together to reconstruct the value $f_{\alpha,\beta}(x)$.

Recently, Servan-Schreiber *et al.* (S&P 2023) investigate the access control problem for DPF; namely, the DPF evaluators can ensure that the DPF dealer is authorized to share the given function with privacy assurance. In this work, we revisit this problem, introducing a new notion called *DPF with constraints*; meanwhile, we identify that there exists a subtle flaw in their privacy definition as well as a soundness issue in one of their proposed schemes due to the lack of validation of the special output value β. Next, we show how to reduce both the storage size of the constraint representation and the server's computational overhead from $O(N)$ to $O(\log N)$, where N is the number of authorized function sets. In addition, we show how to achieve fine-grained private access control, that is, the wildcard-style constraint for the choice of the special output β. Our benchmarks show that the amortized running time of our logarithmic storage scheme is $2\times$ - $3\times$ faster than the state-of-the-art when $N = 2^{15}$. Furthermore, we provide the first impossibility and feasibility results of the *DPF with constraints* where the evaluators do not need to communicate with each other.

*This is an updated version of our work "Fine-grained Policy Constraints for Distributed Point Function", which can be found on IACR ePrint Report 2023/1672; please see [18]. Compared with [18], this version has been significantly improved. We revised and improved the definition of *DPF with constraints*; we added new impossibility and feasibility results of the DPF with constraints where the evaluators do not need to communicate with each other; finally, to improve the presentation and make the results more readable, we removed the part of the DPF with attribute-based constraints.

Contents

1 Introduction

Function Secret Sharing (FSS) is initially introduced by Boyle *et al.* [6]. In FSS, a dealer is allowed to split a secret function f into multiple *succinctly described* function-shares $\{f^{(i)}\}_{i\in\mathbb{Z}_n}$, and any strict subset of the function-shares reveals nothing about the function f. Here, each function-share $f^{(i)}$ can be *individually* evaluated on the common input x, and these evaluation results $\{f^{(i)}(x)\}_{i\in\mathbb{Z}_n}$ can then be *merged together* so that the value $f(x)$ is reconstructed.

Distributed point function. Distributed Point Function (DPF) [13] is arguably the most widely used special case of FSS, where the secret function f is a *point function*. Concretely, a point function f can be defined as follows:

$$f(x) = \begin{cases} \beta \text{ if } x = \alpha \\ 0 \text{ otherwise} \end{cases}$$

We refer to α and β as the *special input* and *special output* of the function f, respectively. For convenience, we often write the point function f as $f_{\alpha,\beta}$.

Utilizing efficient DPF constructions as building blocks, a variety of important applications for private database access have been developed, such as distributed ORAM [8,11,17], anonymous communication [9,12, 23,28], private telemetry [3], and privacy-preserving machine learning [5,22,24]. Note that, most DPF-based applications above utilize the fact that a DPF can be viewed as a *compressed* additive secret-sharing of a weight-1 vector, which enables private reading/writing in a distributed database.

Distributed point function with constraints. In real-world DPF-based applications, one might want to enforce certain *constraints* on the set of functions that the dealer is allowed to share. For instance, in the aforementioned private database access scenarios, different data entries may belong to different owners; private database access control is necessary to prevent malicious users from accessing or altering unauthorized data items. The deployment of a *DPF with certain constraints* enforces the private access control in a distributed database system. Analogously, in an anonymous communication system [12,23,28], a *DPF with constraints* can be used to enforce certain policy so that only the authorized users are allowed to deposit messages into the corresponding mailbox.

In Fig. 1 below, we illustrate the system overview of private databases with access control based on *DPF with constraints*. The system involves the following entities: (1) the data owner who outsources (encrypted) data to servers and determines the constraint policy, (2) the user (i.e., the DPF dealer) who secretly shares a function to access the database and proves it satisfies the constraint, and (3) the servers (i.e., the DPF evaluators) locally evaluate the function share and obliviously verify the user's access. Note that, the user needs a valid secret key as authorization to pass the verification. There are various techniques for key distribution, e.g., anonymous communication channels. We note that the key distribution mechanism is orthogonal to the constrained DPF. For simplicity, in this paper, we assume there exists a trusted authority to issue the secret keys.

Initial results by Servan-Schreiber et al.'s, and security concerns. Very recently, Servan-Schreiber *et al.* [27][1] initiate the study on *DPF with constraints*, which they call "Private Access Control Lists (PACLs) for DPF." In particular, the most efficient scheme (cf. the VDPF-PACL) in [27] is based on verifiable DPF [3,10]. Note that, the verifiable DPF is a special type of DPF that additionally allows evaluators to check whether the function shares are well-formed. However, we find that there is a soundness issue in their VDPF-PACL scheme; see more explanation below.

Consider $\mathbb{G} := \mathbb{Z}_p^*$ as a group with generator g, and a function family $\mathcal{F}$ of point functions $f : \mathcal{D} \to \mathbb{Z}_p$. For each $x \in \mathcal{D}$, there is a constraint configuration assigned for the set of point functions $\{f_{x,v}\}_{v\in\mathbb{Z}_p}$. More specifically, for each $x \in \mathcal{D}$, a pair of keys $(\mathsf{vk}_x, \mathsf{sk}_x)$ is generated, where $g^{\mathsf{sk}_x} = \mathsf{vk}_x$. The verification key vk_x is called the constraint configuration and is public to all participants. The secret key sk_x is held by the authorized dealer. Now, for any point function $f_{\alpha,\beta} \in \mathcal{F}$, upon receiving function-shares $\{f^{(i)}_{\alpha,\beta}\}_{i\in\mathbb{Z}_n}$ from a dealer, each evaluator, say P_i, computes $\mathsf{vk}^{(i)} := \sum_{x\in\mathcal{D}} f^{(i)}_{\alpha,\beta}(x) \cdot \mathsf{vk}_x$ over $\mathbb{Z}_p$. Note that here $\sum_{i\in\mathbb{Z}_n} \mathsf{vk}^{(i)} = \mathsf{vk}_\alpha \cdot \beta$ over $\mathbb{Z}_p$. In [27], the authors assume $\beta = 1$, and the evaluators can jointly verify if the dealer knows the discrete logarithm of $\mathsf{vk}_\alpha \cdot \beta$ to the base g. Although they claim that any user without knowledge of sk_α

[1] Note, the updated version of [27] can be found on IACR ePrint; please see [26].

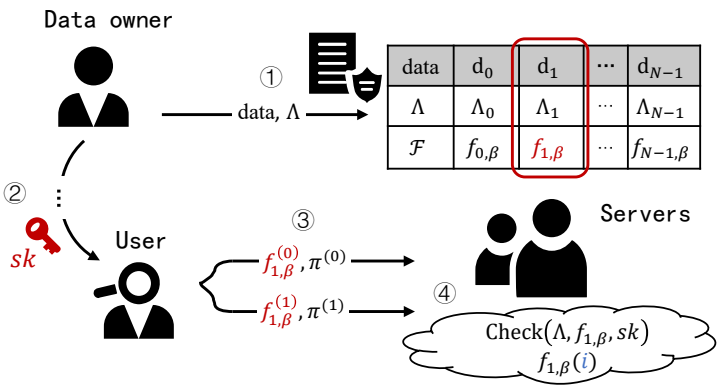

Figure 1: System overview of private databases with access control based on constrained DPF: ① the data owner outsources (encrypted) data with constraint configurations to servers; ② distribute secret keys; ③ the user holds the secret key for $f_{1,\beta}$ accesses d_1 by distributing secret shares of $f_{1,\beta}$ and proof shares to servers; ④ the servers jointly check whether this access satisfies the constraint and evaluate $f_{1,\beta}(i)$ on any common input i.

cannot pass the verification. Unfortunately, the VDPF-PACL scheme fails to validate $\beta = 1$; any malicious adversary is able to bypass the verification by setting $\beta := \mathsf{vk}_\alpha / g^r$ for any forged secret key $\mathsf{sk}^*_\alpha = r$.

Achieving fine-grained access control. On the other hand, the data owner might require more fine-grained database access control than those supported in [26, 27]. In some application scenarios, the data owner may want to also constrain the choice of the special output β. For instance, a landlord provides a template lease; the tenant is only permitted to modify the lease term, and all other terms are fixed. Unfortunately, none of the schemes in [26, 27] supports DPF constraints w.r.t. β.

Achieving better performance. In addition to enabling fine-grained private database access control, all previous approaches require that each evaluator stores at least N verification keys $\{\mathsf{vk}_{x_i}\}_{i\in\mathbb{Z}_N}$ for N constraint configurations assigned for distinct input values $\{x_i \in \mathcal{D}\}_{i\in\mathbb{Z}_N}$. Typically, the evaluators jointly compute $\{f_{\alpha,\beta}(x_i)\}_{i\in\mathbb{Z}_N}$ in the shared form, which is then used to perform the 1-out-of-N configuration selection to obtain vk_α. This results in $O(N)$ storage and $O(N)$ selection computation complexity. We observe that such selection step becomes a performance bottleneck of CDPF when N is large. We are wondering whether there exists a more efficient solution for the key selection.

In this work, we focus on 2-party *DPF with constraints*, or *constrained DPF* (CDPF). To the best of our knowledge, the existing multi-party DPF systems are less efficient [6, 9, 23]. Our constructions can be generalized to multi-party cases as in [27].

Our contributions. In this paper, we systematically investigate the problem of DPF with constraints. Our contributions can be summarized as follows.

New security definition. First, we propose a formal definition for DPF with constraints (CDPF), which can be viewed as an extension of verifiable DPF [10]. For a verifiable DPF, the evaluators can only ensure the shared function is a valid point function; whereas, in CDPF, the evaluators can also obliviously verify whether the dealer is authorized to share this point function. Notice that previous works [12, 23, 27, 28] adopt a stand-alone security definition for constraint mechanism, as their schemes use DPF in a blackbox fashion. Arguably, our definition could enable more efficient CDPF schemes by integrating the DPF with its constraint mechanism in the syntax and security definition.

We note that the privacy definition in [27] is problematic in several aspects. Firstly, their definition did not capture the rushing adversary model; namely, the simulator is given the adversary's message as input. Furthermore, in a real-world application scenario, the PACL in [27] shall be used in conjunction with the underlying DPF schemes; in particular, some algorithm of the PACL takes the function shares generated by DPF as its input. However, in the privacy definition in [27], the view of evaluator does not include the function share. It results in a trivial simulator (cf., the proof of Theorem 3 in [27]) who simulates the audit

token independent of function share. In Appendix B, we provide a simple counterexample to illustrate this subtle problem.

Breaking and fixing the scheme in [27]. As mentioned before, the VDPF-PACL scheme in [27] is insecure against a malicious user. We provide a formal description of our attack in Sec. 4. To fix this issue, a straightforward way is to fix β to some public value, say 1, and validate $\beta = 1$ during the verification phase. Yet, sometimes the dealer needs to set β to a specific private value that is unknown to the evaluators. In this work, we propose an alternative solution using verifiable DPF with auxiliary output. The auxiliary output is independent of the original (secret) point function, thus it can be pre-fixed regardless the value of β. Concretely, given a point function $f_{\alpha,\beta} : \mathcal{D} \to \mathbb{Z}_p$, we transform it to a new point function $f_{\alpha,(\beta,1)} : \mathcal{D} \to \mathbb{Z}_p \times \mathbb{Z}_p$ as follows:

$$f_{\alpha,(\beta,1)}(x) = \begin{cases} (\beta, 1) \text{ if } x = \alpha \\ (0, 0) \text{ otherwise} \end{cases}$$

For each input $x \in \mathcal{D}$, the first coordinate of the new function output is equal to the output of the original $f_{\alpha,\beta}$, and the second coordinate is either 0 or 1 which can be used to defend against our attack on the VDPF-PACL scheme in [27]. In addition, adding an auxiliary output only results in the communication overhead of transmitting a correction word and the computation overhead of a few addition operations. In another word, our fix introduces little overhead.

Improving the performance of the schemes in [26, 27]. We show how to improve the performance of the CDPF schemes in [26, 27] (named DPF-PACL and VDPF-PACL), achieving $O(\log N)$ constraint storage size and $O(\log N)$ selection computation complexity by packing constraint configurations, where N is the number of constraint configurations. Recall that, in [26, 27], for a family of point functions with input domain $\mathbb{Z}_N$, each input $i \in \mathbb{Z}_N$ is mapped to a random key pair $(\mathsf{vk}_i, \mathsf{sk}_i)$, which yields $O(N)$ storage. Let $i_1, i_2, \ldots, i_{\log N}$ be the binary representation of i. Our main idea is to generate $2 \log N$ random keys $((k_{1,0}, k_{1,1}), (k_{2,0}, k_{2,1}), \ldots, (k_{\log N,0}, k_{\log N,1}))$ and then map index i to $\mathsf{vk}_i := \prod_{j=1}^{\log N} k_{j,i_j}$. Then, the 1-out-of N configuration selection is transformed into $\log N$ number of 1-out-of-2 selections.

Note that, there are two challenges to this intuition. First, our new selection method utilizes the binary representation of the special input $\alpha \in \mathbb{Z}_N$ of a point function $f_{\alpha,\beta}$, but the evaluators cannot obliviously bit-decompose the special input α using the (verifiable) DPF technique. To overcome this problem, we present a new primitive of incremental verifiable DPF (IVDPF). For an IVDPF, if the evaluation input set contains α, the layer outputs of the evaluation result form the (scaled) bit decomposition of α (cf. Sec. 5.1, below). The second challenge is that if the secret key is exactly the discrete logarithm of the corresponding verification key, then a malicious user with access to multiple indices may be able to compute the valid secret keys of other unauthorized indices. We introduce bilinear map to address this problem, which blinds the discrete logarithm (cf. Sec. 5.2, below).

New CDPF scheme. We propose a new CDPF scheme that supports ***wildcard-style constraints*** for the special output β. It is handy in the applications of constraint writing, e.g., signature templates. In our *wildcard-style CDPF* scheme, each constraint configuration divides the output of the associated functions into wildcard bits and restricted bits. We say a function satisfies the constraint iff all restricted bits of its output are 0; while the wildcard bits can be chosen arbitrarily. Our construction is based on the fact that the AND operation is distributive over the XOR operation; therefore, the evaluators can locally computes the XOR secret shares of the AND result between the XOR-shared function output and the restraint string (cf. Sec. 6).

Better performance. Our schemes improve upon the state-of-the-art in terms of both functionality and performance. In Sec. 7, we show that our wildcard-style CDPF scheme is much more efficient than PACL schemes in [26, 27]. In addition, when the number of constraint configurations is large, our IVDPF-CDPF significantly improves the performance compared to the VDPF-PACL scheme in [26, 27]. For instance, with 2^{15} constraint configurations, the verification speed of our IVDPF-CDPF is 2.5× faster than the DPF-PACL in [26, 27] and 2× faster than the VDPF-PACL in [26, 27].

Removing the interaction between CDPF evaluators: Impossibility and feasibility. Notice that all our constructions (as well as all existing constructions) require the evaluators to have (at least) one round of interaction between each other, before the decision is made. A natural question is whether there exists a

CDPF scheme where the evaluators could make a decision without any interaction between each other. In Sec. 8, we demonstrate two results: (1) we show in the *plain model*[2], it is impossible to have such CDPF schemes; (2) using certain setups, it is feasible to construct such schemes.

For the impossibility result, intuitively, suppose there exists a secure CDPF scheme that supports verification via non-interactive evaluators. That is, the verification result only depends on a single piece of function share. The malicious dealer can invoke the simulator to produce a valid function share for an unauthorized function, which violates the soundness.

For the feasibility result, we show how to construct such a CDPF scheme in the NIZK and PKI hybrid model, assuming all parties are connected by a broadcast channel. We let the dealer broadcast the encrypted function shares and the audit tokens (needed for the evaluator to make a decision). In addition, the dealer provides an NIZK proof to prove the validity of these audit tokens. Finally, each evaluator can locally check the audit tokens and the NIZK proof to make a verification decision. Furthermore, we propose a more efficient construction in the preprocessing model. Although this scheme has a drawback that it cannot ensure the consensus between evaluators, it is secure enough for private information retrieval applications [16, 29]. Intuitively, the drawback can be addressed by re-randomizing the responses.

Related work. Recent anonymous communication systems [9,12,23,28] utilize DPF to enable users privately write messages. To prevent private data against malicious users, some of them design ad-hoc methods of access control for DPF. The "mailbox" system Express [12] assigns a λ-bit virtual address to each mailbox, which is regardless of the actual number of mailboxes. Only users who know the secret virtual address can deposit message into the corresponding mailbox. Its long virtual address causes costly overhead for DPF evaluation, and its verification requires interaction between servers and client. Sabre [28] improves Express by a secret-shared non-interactive proof. It achieves less communication and computation costs, but requires an extra aided-server for auditing the well-formedness of DPF. Spectrum [23] constructs an anonymous broadcasting system. It proposes a new access control mechanism via secret-shared Carter-Wegman MAC [30]. That is, servers hold a secret-shared MAC tag for each broadcast channel, and only allow the client who knows the valid MAC tag to write messages to the corresponding channel via DPF.

Servan-Schreiber *et al.* [26, 27] abstract the notion of PACLs for FSS from concrete applications. They improve Spectrum's technique to more efficient PACLs for DPF, and propose a general PACL for P/poly FSS. However, their general PACL relies on costly secret-shared non-interactive proofs and is not quit efficient in practice.

Organization. Preliminaries including notations, background concepts, and assumptions can be found in Sec. 2. The formal definition of CDPF can be found in Sec. 3. The VDPF-PACL scheme in [27] is revisited in Sec. 4. A fast CDPF with logarithmic storage is presented in Sec. 5. Our wildcard-style CDPF is presented in Sec. 6. Our implementation and benchmark are shown in Sec. 7. Our impossibility and feasibility results of CDPF with non-interactive evaluators are presented in Sec. 8.

In addition, we recap the definition of VDPF [10] in Appendix A. We recap the definition of PACL [27] and discuss its several problems in Appendix B. We define the security properties of CDPF for a special case in Appendix C. We present a CDPF scheme with non-interactive evaluators in the preprocessing model in Appendix D. Finally, we give some proofs omitted from the main paper in Appendix E.

2 Preliminaries

Notations. Let λ denote the security parameter. Let p be a prime number. For $x \in \mathbb{Z}_p$, let $[\![x]\!]$ denote the additively secret sharing of the value x. Here, $[\![x]\!] \leftarrow \mathsf{AddShare}_{\mathbb{Z}_p,n}(x)$ stands for generating n additive shares of x, where $[\![x]\!] := \{x^{(0)}, \ldots, x^{(n-1)}\}$, $x^{(i)} \in \mathbb{Z}_p$ for all $i \in \mathbb{Z}_n$, and $x = \sum_{i\in\mathbb{Z}_n} x^{(i)}$ over $\mathbb{Z}_p$. $([\![x]\!], [\![y]\!], [\![z]\!]) \leftarrow \mathsf{Beaver}_{\mathbb{Z}_p,n}$ stands for generating a beaver triple, i.e., generating n additive shares of $x, y, z \leftarrow \mathbb{Z}_p$, such that $x \cdot y = z$. Let $\mathbb{G}$ be a circle group. For $x \in \mathbb{G}$, let $\langle x \rangle$ denote the multiplicatively secret sharing of the value x. Here, $\langle x \rangle \leftarrow \mathsf{MulShare}_{\mathbb{G},n}(x)$ stands for generating n multiplicative shares of x, where $\langle x \rangle := \{x^{(0)}, \ldots, x^{(n-1)}\}$ and $x = \prod_{i\in\mathbb{Z}_n} x^{(i)}$ over $\mathbb{G}$.

[2]The plain model refers to a setting where schemes do not rely on any form of setup, and their security is based on computational hardness assumptions.

Access Structure [2]. Let $\mathcal{P} := \{\mathsf{P}_1, \mathsf{P}_2, \ldots, \mathsf{P}_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\mathcal{P}}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$. An access structure $\mathbb{A}$ (respectively, monotone access structure) is a collection (respectively, monotone collection) of non-empty subsets of $\mathcal{P}$, i.e., $\mathbb{A} \subseteq 2^{\mathcal{P}} \setminus \{\emptyset\}$. We call the sets in $\mathbb{A}$, the *authorized sets*, and the ones not in $\mathbb{A}$, the *unauthorized sets*.

Function Secret Sharing. Function Secret Sharing, introduced by Boyle *et al.* [6], provides a way for a dealer to additively secret-share a function among evaluators. An FSS scheme consists of (at least) two algorithms $\mathsf{Gen}, \mathsf{Eval}$. The dealer generates secret shares of a function $f : \mathcal{D} \to \mathbb{G}$ by Gen. Using the function shares, evaluators can locally execute Eval to produce additive shares of $f(x)$ for any $x \in \mathcal{D}$, without learning information of f. DPF [13] is a useful FSS instance for the point function family.

Recently, a new notion, called *verifiable* DPF (VDPF), has been proposed to ensure the integrity of DPF against malicious dealer [3, 10]. On top of the conventional DPF security guarantees, a VDPF enables the evaluators to check whether the dealer's inputs are well-formed point function. Informally, a VDPF consists of three PPT algorithms $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Verify})$. The share generation algorithm Gen produces the secret shares of the point function $f_{\alpha,\beta}$. The verifiable evaluation algorithm Eval evaluates the function share over a set of inputs X, and produces the additive shares of the function outputs and a token. The verification function Verify takes input as tokens, and outputs 1 iff the function outputs have at most one non-zero item, which means that the reconstructed function is a well-formed point function. The formal VDPF definition can be found in Appendix A.

Bilinear maps and the BDH assumption. Let $\mathbb{G}, \mathbb{G}_t$ be two multiplicative cyclic groups of prime order p, and g be a generator of $\mathbb{G}$. We say e is a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$, if it has two properties:

- Bilinearity: $e(u^a, v^b) = e(u, v)^{ab}$, $\forall u, v \in \mathbb{G}, a, b \in \mathbb{Z}_p$.
- Non-degeneracy: $e(g, g) \neq 1$.

The decisional Bilinear Diffie-Hellman (BDH) assumption [4, 15] is that, given a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$, for $a, b, c, z \leftarrow \mathbb{Z}_p$, no PPT adversary can distinguish the tuple $(g^a, g^b, g^c, e(g, g)^{abc})$ and $(g^a, g^b, g^c, e(g, g)^z)$ with more than a negligible advantage.

XOR-collision resistant hash functions. Following the definition in [10], we say a function family $\mathcal{H}$ is XOR-collision resistant if no PPT adversary given a randomly sampled $h \leftarrow \mathcal{H}$ finds four values x_0, x_1, x_2, x_3 such that $(x_0, x_1) \neq (x_2, x_3)$, $(x_0, x_1) \neq (x_3, x_2)$ and $h(x_0) \oplus h(x_1) = h(x_2) \oplus h(x_3) \neq 0$ with more than a negligible probability.

3 DPF with Constraints

In this section, we present a formal definition of *DPF with constraints* (CDPF), which can be viewed as a generalization of the *Private Access Control Lists (PACL)* for DPF proposed in [26, 27]. A bit more concretely, we take two steps to enforce constraints for DPF.

- *Defining the constraints.* Let $\mathcal{F}$ be a function family of point functions. Let $\{\mathcal{Q}_i\}_{i \in \mathbb{Z}_N}$ be N number of disjoint subsets of authorized functions, that is, for any $i \neq j$, it holds that $\mathcal{Q}_i \cap \mathcal{Q}_j = \emptyset$. Now, for each distinct authorized subset $\mathcal{Q}_i \subseteq \mathcal{F}$, a constraint configuration Λ_i can be defined. Then for function family $\mathcal{F}$, a constraint configuration list $\Lambda := \{\Lambda_i\}$ is defined.
- *Enabling the verification of the constraints.* Next, an efficiently computable predicate Check is introduced to verify if a constraint configuration is valid. Concretely, given a function f and a secret key sk, we have $\mathsf{Check}(\Lambda, f, \mathsf{sk}) = 1$ iff f belongs to an authorized subset, say $\mathcal{Q}_i$, and (Λ_i, sk) belongs to an efficiently decidable binary relation R. In particular, each constraint configuration Λ_i may be a set of multiple entries and the secret key sk corresponding to any of entries satisfies $R(\Lambda_i, \mathsf{sk}) = 1$. Throughout, we denote the number of entries as ℓ.

In this paper, we focus on 2-party CDPF (i.e., DPF with constraints), where several roles, including (i) *two* evaluators (also called servers) along with (ii) a data owner, and (iii) a dealer (also called user), are involved; please also see Fig. 1. Very briefly, the two evaluators $\mathsf{P}_0, \mathsf{P}_1$ can securely attest $\mathsf{Check} = 1$ over the secret-shared function from the dealer D. Furthermore, similar to the verifiable DPF [10], we consider the malicious security for a single corrupted party. That is, the deader and evaluators are non-colluding. Formally, we define the 2-party CDPF as follows:

Definition 1 (2-party DPF with constraints)**.** *Let $\mathcal{F}$ be a family of point functions $f : \mathcal{D} \to \mathbb{G}$ where $\mathbb{G}$ is an Abelian group*[3]*, and N be an integer in $\mathsf{poly}(\lambda)$ such that $N \leq |\mathcal{D}|$. A 2-party constrained distributed point function for function family $\mathcal{F}$, consists of five* PPT *algorithms* $(\mathsf{Constraint}, \mathsf{SKGen}, \mathsf{FGen}, \mathsf{VEval}, \mathsf{Verify})$ *as follows:*

- (Λ, msk) *or* $\perp \leftarrow \mathsf{Constraint}(1^\lambda, \{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N})$ *is the constraint generation algorithm that takes input as the security parameter 1^λ and the set of function sets $\mathcal{Q}_i \subseteq \mathcal{F}$ with the access structure $\mathbb{A}_i$. It outputs a tuple of a constraint configuration list $\Lambda := \{\Lambda_i\}_{i\in\mathbb{Z}_N}$ and a master secret key msk, or a symbol $\perp$ for error.*
- $\mathsf{sk} \leftarrow \mathsf{SKGen}(1^\lambda, \mathsf{id}, \mathsf{msk})$ *is the secret key generation algorithm*[4] *that takes input as the security parameter 1^λ, an identity description id and a master secret key msk. It outputs a secret key sk.*
- $((f^{(0)}, \pi^{(0)}), (f^{(1)}, \pi^{(1)})) \leftarrow \mathsf{FGen}(\Lambda, f, \mathsf{sk})$ *is the share generation algorithm that takes input as a constraint configuration list Λ, function $f \in \mathcal{F}$ and a secret key sk. It outputs a pair of function shares with proof shares.*
- $(\{y_x^{(b)}\}_{x\in X}, \tau^{(b)}) \leftarrow \mathsf{VEval}(b, \Lambda, f^{(b)}, \pi^{(b)}, X)$ *is the verifiable evaluation algorithm that takes input as an index $b \in \{0, 1\}$, a constraint configuration list Λ, a function share $f^{(b)}$, a proof share $\pi^{(b)}$, and a set of function inputs $X \subseteq \mathcal{D}$. It outputs a tuple of values. The first set of values $\{y_x^{(b)}\}$ are DPF outputs, and the second item is an audit token $\tau^{(b)}$.*
- 1 *or* $0 \leftarrow \mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$ *is the verification algorithm that takes input as two tokens. It outputs* 1 *or* 0.

The above algorithms implicitly take group descriptions as their input. For readability, we omit them in the syntax when they are clear in the context. We say a constrained distributed point function $\mathsf{CDPF} = (\mathsf{Constraint}, \mathsf{SKGen}, \mathsf{FGen}, \mathsf{VEval}, \mathsf{Verify})$ *for function family $\mathcal{F}$ is secure if it satisfies three properties as follows:*

- ***Completeness.*** *For any $\{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N}$, $L \leftarrow \mathsf{Constraint}(1^\lambda, \{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N})$ such that $L \neq \perp$ and parse $L := (\Lambda, \mathsf{msk})$, any $i \in \mathbb{Z}_N$, any $f \in \mathcal{Q}_i$, any $\mathsf{sk} \leftarrow \mathsf{SKGen}(1^\lambda, \mathsf{id}, \mathsf{msk})$ where id belongs to $\mathbb{A}_i$, and any $X \subseteq \mathcal{D}$, we have*

$$\Pr\left[\begin{array}{l} ((f^{(0)}, \pi^{(0)}), (f^{(1)}, \pi^{(1)})) \leftarrow \mathsf{FGen}(\Lambda, f, \mathsf{sk}); \\ (\{y_x^{(0)}\}_{x\in X}, \tau^{(0)}) \leftarrow \mathsf{VEval}(0, \Lambda, f^{(0)}, \pi^{(0)}, X); \\ (\{y_x^{(1)}\}_{x\in X}, \tau^{(1)}) \leftarrow \mathsf{VEval}(1, \Lambda, f^{(1)}, \pi^{(1)}, X) : \\ (\forall x \in X,\ y_x^{(0)} + y_x^{(1)} = f(x)) \wedge \\ (\mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1) \end{array}\right] = 1$$

- ***Privacy.*** *For a constraint configuration list Λ, a function $f \in \mathcal{F}$, a secret key sk and a set of function inputs $X \subseteq \mathcal{D}$, define the distribution of the view of P_b as*

$$\mathsf{View}_{\mathsf{CDPF}}(b, \Lambda, f, \mathsf{sk}, X) := \{(f^{(b)}, \pi^{(b)}, \tau^{(1-b)})\}_\lambda$$

where $((f^{(0)}, \pi^{(0)}), (f^{(1)}, \pi^{(1)})) \leftarrow \mathsf{FGen}(\Lambda, f, \mathsf{sk})$ and $(Y, \tau^{(1-b)}) \leftarrow \mathsf{VEval}(1 - b, \Lambda, f^{(1-b)}, \pi^{(1-b)}, X)$. There exists a PPT *simulator Sim such that for any Λ, msk generated from $\mathsf{Constraint}(1^\lambda, \{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N})$ for any $\{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N}$, any function f from any $\mathcal{Q}_i$, any secret key $\mathsf{sk} \leftarrow \mathsf{SKGen}(1^\lambda, \mathsf{id}, \mathsf{msk})$ where id belongs to $\mathbb{A}_i$, any $X \subseteq \mathcal{D}$, and any $b \in \{0, 1\}$, the following two distributions are computationally indistinguishable:*

$$\mathsf{View}_{\mathsf{CDPF}}(b, \Lambda, f, \mathsf{sk}, X) \approx_c \mathsf{Sim}(1^\lambda, b, \Lambda, X)$$

- ***Soundness.*** *For any* PPT *adversary $\mathcal{A}$, it holds that*

$$\Pr[\mathbf{G}^{\mathsf{sound}}_{\mathsf{CDPF},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where the game $\mathbf{G}^{\mathsf{sound}}_{\mathsf{CDPF},\mathcal{A}}(\lambda)$ is depicted in Fig. 2.

Remark 1. *In this section, we define the security properties of CDPF for the normal case where the verification result of P_b depends on both $\tau^{(0)}$ and $\tau^{(1)}$. For completeness, we define different security properties for the special case where the verification result of P_b only depends on $\tau^{(b)}$, which can be found in Appendix C.*

[3]Note that the order of $\mathbb{G}$ is not important in DPF and can even be 2.

[4]The representation of id depends on the concrete system. For example, in PACL [26, 27] exact identities are considered.

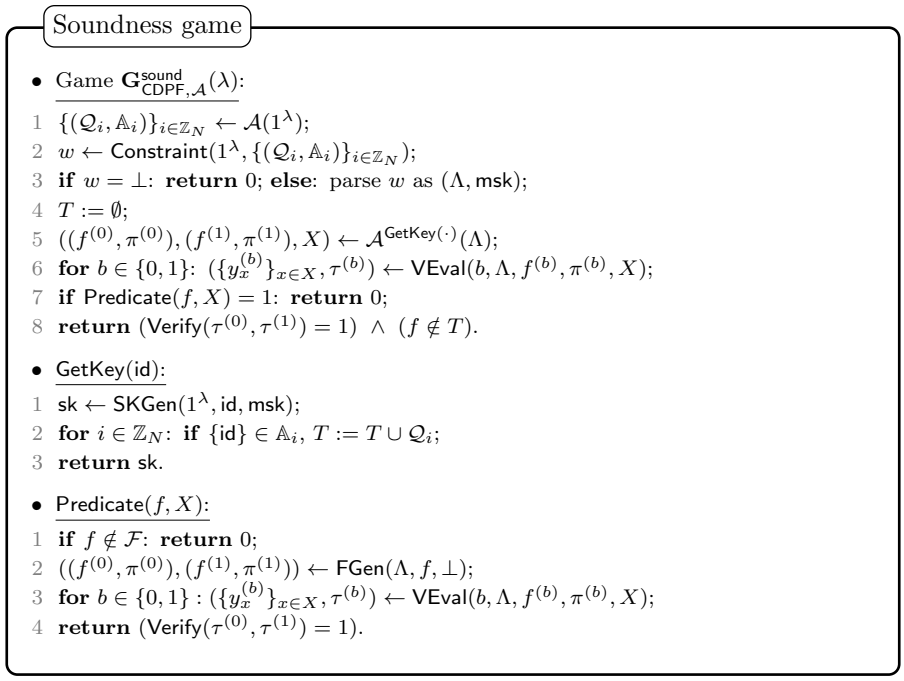

Figure 2: Soundness game for constrained distributed point function $\mathsf{CDPF} = (\mathsf{Constraint}, \mathsf{SKGen}, \mathsf{FGen}, \mathsf{VEval}, \mathsf{Verify})$ for function family $\mathcal{F}$.

Remark 2. *We note that the privacy definition of [27] is problematic and does not capture the concept of rushing adversaries. Concretely, in the privacy definition in [27], the view of evaluator does not include the function share. It results in a trivial simulator (cf. the proof of Theorem 3 in [27]) who simulates the token independent of function share. In Appendix B, we provide more details; in particular, we provide a counterexample to illustrate this subtle problem.*

4 Revisiting VDPF-PACL in [27]

The recent work [27] presents PACL schemes for the class of (verifiable) DPFs. In this section, we show their VDPF-PACL is not sound for the adversarially chosen β.

The VDPF-PACL scheme. Consider $\mathbb{G} := \mathbb{Z}_p^*$ as a group with order $p-1$ and generator g in which the discrete logarithm problem is assumed to be computationally intractable.[5] Without loss of generality, let $\ell = 1$ and focus on 2-party VDPF. For a family $\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{Z}_p\}$ of point functions and $N := 2^n$, the VDPF-PACL initiates constants $\Lambda := \{\mathsf{vk}_i\}_{i\in\mathbb{Z}_N}$ by KeyGen algorithm:

- $\mathsf{KeyGen}(1^\lambda, f_{i,\beta} \in \mathcal{F})$:

1 $\mathsf{sk}_i \leftarrow \mathbb{Z}_{p-1}$, $\mathsf{vk}_i := g^{\mathsf{sk}_i}$;
2 **return** $(\mathsf{vk}_i, \mathsf{sk}_i)$.

In [27], the VDPF-PACL uses *Schnorr Proof over Secret Shares* (SPoSS) as a building block. SPoSS is a discrete-logarithm zero-knowledge proof-of-knowledge over an additively secret-shared element. More specifically, it consists three PPT algorithms: (1) $\mathsf{SPoSS.Prove}(x) \to (\pi^{(0)}, \pi^{(1)})$ takes the discrete logarithm x of w base g as input, and outputs the secret sharing of a non-interactive zero-knowledge proof; (2) $\mathsf{SPoSS.Audit}(b, w^{(b)}, \pi^{(b)}) \to \tau^{(b)}$ takes the additive share of w and the SPoSS proof share as input, and outputs a verification token; (3) $\mathsf{SPoSS.Verify}(\tau^{(0)}, \tau^{(1)})$ takes the tokens of two evaluators as input, and outputs 1 if and only if $w^{(0)} + w^{(1)} = g^x$ over the field $\mathbb{Z}_p$.

[5] Refer to Chapter 10 of [19] for how to pick a suitable p. For example, the largest prime dividing $p-1$ is sufficiently large.

Here, we briefly describe the access process of VDPF-PACL based on SPoSS. First of all, the user chooses an point function $f_{\alpha,\beta} \in \mathcal{F}$, and secret-shares it to two servers using the VDPF technique. Meanwhile, the user plays the role of the prover to provide a SPoSS proof of the secret key $\mathsf{sk} \in \mathbb{Z}_{p-1}$ for the secret-shared $[\![g^{\mathsf{sk}}]\!]$ as follows:

- $\mathsf{Prove}(f_{\alpha,\beta}, \mathsf{sk})$:

1 $(\pi^{(0)}, \pi^{(1)}) \leftarrow \mathsf{SPoSS.Prove}(\mathsf{sk})$;
2 **return** $(\pi^{(0)}, \pi^{(1)})$.

Upon receiving the proof shares from the user, each server $P_b \in \{P_0, P_1\}$ as a verifier obliviously selects the target verification key by the function share $f^{(b)}_{\alpha,\beta}$ and audits the SPoSS proof using Audit algorithm:

- $\mathsf{Audit}(b, \Lambda, f^{(b)}_{\alpha,\beta}, \pi^{(b)})$:

1 Parse $\Lambda := \{\mathsf{vk}_i\}_{i\in\mathbb{Z}_N}$;
2 $(\{y^{(b)}_i\}_{i\in\mathbb{Z}_N}, \tau^{(b)}_0) \leftarrow \mathsf{VDPF.Eval}(b, f^{(b)}_{\alpha,\beta}, \mathbb{Z}_N)$;
3 $w^{(b)} := \sum_{i=0}^{N-1} \mathsf{vk}_i \cdot y^{(b)}_i \pmod p$;
4 $\tau^{(b)}_1 \leftarrow \mathsf{SPoSS.Audit}(b, w^{(b)}, \pi^{(b)})$;
5 **return** $\tau^{(b)} := (\tau^{(b)}_0, \tau^{(b)}_1)$.

Finally, two evaluators exchange their audit tokens $\tau^{(0)}, \tau^{(1)}$, and verify the well-formedness of VDPF and the SPoSS proof using Verify algorithm:

- $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$:

1 Parse $\tau^{(0)} := (\tau^{(0)}_0, \tau^{(0)}_1)$, and $\tau^{(1)} := (\tau^{(1)}_0, \tau^{(1)}_1)$;
2 **return** $\mathsf{VDPF.Verify}(\tau^{(0)}_0, \tau^{(1)}_0)$
$\wedge\ \mathsf{SPoSS.Verify}(\tau^{(0)}_1, \tau^{(1)}_1)$.

It is easy to see, Verify outputs 1 if and only if both the VDPF and the SPoSS verification pass.

The description of our attack. In light of the VDPF-PACL, the SPoSS just guarantees that the additively secret-shared $[\![w]\!] := g^{\mathsf{sk}_\alpha} \cdot [\![\beta]\!]$ is equal to $g^{[\![\mathsf{sk}]\!]}$ over the field $\mathbb{Z}_p$. This ignorance of β is vulnerable, and we show how to exploit it. We construct an adversary $\mathcal{A}$ who breaks the soundness of VDPF-PACL:

- $\mathcal{A}(1^\lambda, \Lambda)$:

1 Parse $\Lambda := \{\mathsf{vk}_i\}_{i\in\mathbb{Z}_N}$;
2 $\alpha \leftarrow \{0,1\}^n$, $r \leftarrow \mathbb{Z}_{p-1}$, $\beta := g^r \cdot \mathsf{vk}_\alpha^{-1} \pmod p$;
3 $(f^{(0)}_{\alpha,\beta}, f^{(1)}_{\alpha,\beta}) \leftarrow \mathsf{VDPF.Gen}(1^\lambda, f_{\alpha,\beta})$;
4 $(\pi^{(0)}, \pi^{(1)}) \leftarrow \mathsf{Prove}(\Lambda, f_{\alpha,\beta}, r)$;
5 **return** $((f^{(0)}_{\alpha,\beta}, f^{(1)}_{\alpha,\beta}), (\pi^{(0)}, \pi^{(1)}))$.

Using the VDPF keys $(f^{(0)}_{\alpha,\beta}, f^{(1)}_{\alpha,\beta})$, two evaluators obliviously obtain a scaled verification key $[\![w]\!] := \Lambda_\alpha \cdot [\![\beta]\!] := [\![g^r]\!]$, and thus the proof π for x over w is verified by SPoSS. In addition, since $(f^{(0)}_{\alpha,\beta}, f^{(1)}_{\alpha,\beta})$ encodes a point function, the VDPF verification passes. Therefore, the algorithm Verify outputs 1 with the probability 1. In a word, $\mathcal{A}$ successfully forges a proof by choosing an appropriate $\beta \in \mathbb{Z}_p$.

Our fix. This soundness issue is based on the fact that the private value β of the point function is adversarially chosen from $\mathbb{Z}_p$, and it scales the verification key vk_α. Naively, if the evaluators additionally check $\beta = 1$, our attack is fixed. However, in many applications (e.g. the mailbox system [12]), one may need to use the customized β, such as anonymous communication. To overcome this problem, we introduce

an auxiliary output. In particular, for any point function $f_{\alpha,\beta} : \{0,1\}^n \to \mathbb{Z}_p$, we can construct a new point function $f_{\alpha,(\beta,1)} : \{0,1\}^n \to \mathbb{Z}_p^2$, where the tuple $(\beta, 1)$ denotes the new special output in the range $\mathbb{Z}_p^2$, and its second item 1 is named the special auxiliary output. We let the dealer secret-share a new point function $f_{\alpha,(\beta,1)}$ via VDPF to be evaluated. After that, the evaluators use the auxiliary outputs to select the target verification key, and then jointly check if the special auxiliary output is equal to 1. Concretely, we adapted the Audit and Verify of the VDPF-PACL as follows:

- $\mathsf{Audit}(b, \Lambda, f^{(b)}_{\alpha,(\beta,1)}, \pi^{(b)})$:

1 $(\{y_i^{(b)}\}_{i\in\mathbb{Z}_N}, \tau_0^{(b)}) \leftarrow \mathsf{VDPF.Eval}(b, f^{(b)}_{\alpha,(\beta,1)}, \mathbb{Z}_N)$;

2 **for** $i \in \mathbb{Z}_N$: Parse $y_i^{(b)} := (y_{i,0}^{(b)}, y_{i,1}^{(b)}) \in \mathbb{Z}_p^2$;

3 $w^{(b)} := \sum_{i=0}^{N-1} \Lambda_i \cdot y_{i,1}^{(b)}$;

4 $\tau_1^{(b)} \leftarrow \mathsf{SPoSS.Audit}(b, w^{(b)}, \pi^{(b)})$;

5 $\tau_2^{(b)} := H(b + (-1)^b \cdot \sum y_{i,1}^{(b)})$;

6 **return** $\tau^{(b)} := (\tau_0^{(b)}, \tau_1^{(b)}, \tau_2^{(b)})$.

- $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$:

1 Parse $\tau^{(0)} := (\tau_0^{(0)}, \tau_1^{(0)}, \tau_2^{(0)})$;

2 Parse $\tau^{(1)} := (\tau_0^{(1)}, \tau_1^{(1)}, \tau_2^{(1)})$;

3 **return** $\mathsf{VDPF.Verify}(\tau_0^{(0)}, \tau_0^{(1)})$
$\wedge\ \mathsf{SPoSS.Verify}(\tau_1^{(0)}, \tau_1^{(1)})\ \wedge\ \tau_2^{(0)} = \tau_2^{(1)}$.

where $H : \{0,1\}^* \to \{0,1\}^\lambda$ is a collision-resistant hash function. We note that, the equality of $\tau_2^{(0)} = \tau_2^{(1)}$ restricts the non-zero element in $\{y_{i,1}^{(0)} + y_{i,1}^{(1)}\}$ is equal to 1, which ensures $[\![w]\!]$ is always a valid (i.e., non-scaled) verification key in Λ. Therefore, the adversary cannot forge a proof π by scaling verification key.

5 CDPF with Logarithmic Storage

We observe that, the existing CDPF schemes (e.g. PACLs in [26, 27]) require that evaluators store at least N verification keys for N constraint configurations. This is because that their underlying DPF expands to a vector of length N to indicate the 1-out-of-N configuration selection. Furthermore, the selection step is the performance bottleneck when N is large. In this section, we show how to construct a CDPF scheme with logarithmic constraint storage size and logarithmic selection computation complexity.

For simplicity, we focus on the case where each constraint configuration has only one entry, i.e., $\ell = 1$, and our scheme generalizes to arbitrary ℓ. Intuitively, we use $2 \log N$ public keys to represent N verification keys as constraint configurations. Let $\mathbb{G}$ be a group with the generator g and $\alpha_1, \ldots, \alpha_{\log N}$ be the bits of α. We set the public keys to $(k_{i,0} := g^{r_{i,0}}, k_{i,1} := g^{r_{i,1}})_{i\in[\log N]}$, and select the α-th verification key by $\mathsf{vk}_\alpha := \prod_{i\in[\log N]} k_{i,\alpha_i}$. Only one who has the discrete-logarithm knowledge of vk_α can produce a valid proof for α-th verification key.

As we already discussed in the Sec. 1, there are two challenges to this intuition: (1) the evaluators are required to obliviously bit-decompose the special input α for configuration selection; (2) the secret key should not be the discrete logarithm of the corresponding verification key. To overcome these problems, we present a verifiable DPF scheme with layer outputs, called incremental VDPF, and introduce bilinear map to blind the discrete logarithm, respectively.

5.1 Incremental VDPF

For a point function $f_{\alpha,\beta}$ with input domain $\{0,1\}^n$, consider a batch evaluation of $f_{\alpha,\beta}$ on any set of inputs $X \subseteq \{0,1\}^n$. In a standard (verifiable) DPF scheme, given the function shares of $f_{\alpha,\beta}$, the evaluators obtain secret-shared function outputs (i.e., a vector of dimension $|X|$ that is non-zero at most a single point) after the batch evaluation. In an incremental VDPF scheme, the evaluators additionally obtain secret-shared layer outputs $\{z_{i,0}, z_{i,1}\}_{i\in[n]}$ such that for any $i \in [n], b \in \{0,1\}$,

$$z_{i,b} = \begin{cases} \beta'_i \text{ if } \alpha_i = b \ \wedge \ \alpha_{[1,i]} \in X_{[1,i]} \\ 0 \text{ otherwise} \end{cases}$$

where α_i denotes the i-th bit of α, $\alpha_{[1,i]}$ denotes the i-bit prefix of α, and $X_{[1,i]}$ denotes the set of i-bit prefixes of all element in X. We refer to $\{\beta'_1, \ldots, \beta'_n\}$ as the special layer outputs. We define the incremental VDPF as follows:

Definition 2 (Incremental VDPF)**.** *Let $\mathcal{F}$ be a function family of point functions $f : \{0,1\}^n \to \mathbb{G}$ where $\mathbb{G}$ is an Abelian group, and $\{\mathbb{G}'_i\}_{i\in[n]}$ be Abelian groups. A 2-party incremental VDPF scheme, parameterized by $\mathcal{F}$ and $\{\mathbb{G}'_i\}_{i\in[n]}$, consists of three PPT algorithms* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Verify})$*:*

- $(f^{(0)}_{\alpha,\beta}, f^{(1)}_{\alpha,\beta}) \leftarrow \mathsf{Gen}(1^\lambda, f_{\alpha,\beta}, \{\beta'_i\}_{i\in[n]})$ *is the share generation algorithm that takes input as a security parameter 1^λ, a point function $f_{\alpha,\beta} \in \mathcal{F}$ and the special layer outputs $\{\beta'_i \in \mathbb{G}'_i\}_{i\in[n]}$. It outputs a pair of IVDPF keys.*
- $(\{y^{(b)}_x\}_{x\in X}, \{z_{i,0}, z_{i,1}\}_{i\in[n]}, \tau^{(b)}) \leftarrow \mathsf{Eval}(b, f^{(b)}_{\alpha,\beta}, X)$ *is the verifiable evaluation algorithm that takes input as an index $b \in \{0,1\}$, an IVDPF key $f^{(b)}_{\alpha,\beta}$ and a set of inputs $X \subseteq \{0,1\}^n$. It outputs a tuple of values. The first set of values are additive shares of $f_{\alpha,\beta}(x), x \in X$. The second set of values are additive shares of layer outputs. The last item is a token.*
- 1 *or* $0 \leftarrow \mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$ *is the verification algorithm that takes input as two tokens. It outputs* 1 *or* 0*.*

A secure IVDPF must satisfy three properties as follows:

Completeness. *For any function $f_{\alpha,\beta} \in \mathcal{F}$, any special layer outputs $\{\beta'_i \in \mathbb{G}_i\}_{i\in[n]}$, any $(f^{(0)}_{\alpha,\beta}, f^{(1)}_{\alpha,\beta}) \leftarrow \mathsf{Gen}(1^\lambda, f_{\alpha,\beta}, \{\beta'_i\}_{i\in[n]})$ and any $X \subseteq \{0,1\}^n$, it holds that*

$$\Pr\left[\begin{array}{l} (Y^{(0)}, Z^{(0)}, \tau^{(0)}) \leftarrow \mathsf{Eval}(0, f^{(0)}_{\alpha,\beta}, X); \\ (Y^{(1)}, Z^{(1)}, \tau^{(1)}) \leftarrow \mathsf{Eval}(1, f^{(1)}_{\alpha,\beta}, X) : \\ \forall x \in X, y^{(0)}_x + y^{(1)}_x = f(x) \ \wedge \\ \forall i \in [n], z^{(0)}_{i,\bar{\alpha}_i} + z^{(1)}_{i,\bar{\alpha}_i} = 0 \ \wedge \\ z^{(0)}_{i,\alpha_i} + z^{(1)}_{i,\alpha_i} = (\alpha_{[1,i]} \in X_{[1,i]}) \cdot \beta'_i \ \wedge \\ \mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1 \end{array}\right] = 1$$

where $Y^{(b)} := \{y^{(b)}_x\}_{x\in X}$, $Z^{(b)} := \{z^{(b)}_{i,0}, z^{(b)}_{i,1}\}_{i\in[n]}$ for $b \in \{0,1\}$, α_i is the i-th bit of α, $\alpha_{[1,i]}$ is the i-bit prefix of α, and $X_{[1,i]}$ is the set of i-bit prefixes of all element in X.

Privacy. *For a point function $f \in \mathcal{F}$, the special layer outputs $B := \{\beta'_i \in \mathbb{G}'_i\}_{i\in[n]}$, and a set of inputs $X \subseteq \{0,1\}^n$, define the view of evaluator P_b $\mathsf{View}_{\mathsf{VDPF}}(b, f, B, X)$ as a probability distribution ensemble $\{(f^{(b)}, \tau^{(1-b)})\}_\lambda$, where the function shares $(f^{(0)}, f^{(1)})$ are generated from $\mathsf{Gen}(1^\lambda, f, B)$, and the audit token $\tau^{(1-b)}$ is computed by $\mathsf{Eval}(1-b, f^{(1-b)}, X)$. There exists a PPT simulator Sim such that for all $f \in \mathcal{F}, B := \{\beta'_i \in \mathbb{G}'_i\}_{i\in[n]}, X \subseteq \{0,1\}^n$, the two distributions are computationally indistinguishable:*

$$\mathsf{View}_{\mathsf{VDPF}}(b, f, B, X) \approx_c \mathsf{Sim}(1^\lambda, b, n, \mathbb{G}, \{\mathbb{G}'_i\}_{i\in[n]}, X)$$

Soundness. *Let $f^{(b)}$ be the (possibly maliciously generated) function-share received by the evaluator P_b. There exists a negligible function* negl *such that for any $X \subseteq \{0,1\}^n$ and $(\{y^{(b)}_x\}_{x\in X}, \{z^{(b)}_{i,0}, z^{(b)}_{i,1}\}_{i\in[n]}, \tau^{(b)}) \leftarrow$*

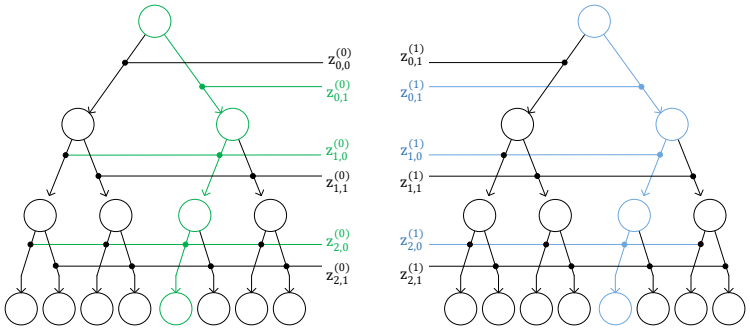

Figure 3: DPF with layer outputs

$\mathsf{Eval}(b, f^{(b)}, X)$ *for* $b \in \{0,1\}$*, it holds that*

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1 : \\ \left|\{x \in X | y_x^{(0)} + y_x^{(1)} \neq 0\}\right| \leq 1 \ \wedge \\ \forall i \in [n], |C_i| \geq 1 \ \wedge \\ \left(\begin{array}{l} \forall x \in X, y_x^{(0)} + y_x^{(1)} = 0 \ \vee \\ \forall i \in [n], x_i \in C_i \end{array}\right) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

where $C_i := \{0,1\}/\{1-j | z_{j,0}^{(0)} + z_{j,0}^{(1)} \neq 0, j \in \{0,1\}\}$ *and* x_i *stands for the* i*-th bit of* x*.*

Our IVDPF construction is inspired by two techniques: the first is called "constant PRG optimization for distributed DPF generation" in [11], and the second is "incremental DPF" in [3]. We below will provide a high-level explanation of our construction ideas. Recap the DPF representation in [7]. For the point function $f_{\alpha,\beta}$, each of the two DPF keys defines a GGM-style binary tree [14] with 2^n leaves, and every node in the tree is labeled by a pseudo-random value. The DPF construction guarantees that, if a node is in the path from root to the α-th leaf, its labels in two trees are independent; otherwise, its labels are identical. Let α_i be the i-th bit of α and $\bar{\alpha}_i := 1 - \alpha_i$. As shown in Fig. 3, for the sums $z_{i,0}^{(b)}, z_{i,1}^{(b)}$ of all left and right children at level i, we have $z_{i,\alpha_i}^{(0)} \neq z_{i,\alpha_i}^{(1)}$ and $z_{i,\bar{\alpha}_i}^{(0)} = z_{i,\bar{\alpha}_i}^{(1)}$. Therefore, we can instruct the i-th layer output by making $z_{i,\alpha_i}^{(0)}, z_{i,\alpha_i}^{(1)}$ reconstruct β_i' and correcting $z_{i,\bar{\alpha}_i}^{(0)}, z_{i,\bar{\alpha}_i}^{(1)}$ to 0.

More specifically, our IVDPF construction is depicted in Fig. 4. In particular, the label of each node includes a pseudo-random seed $\tilde{s}^{(b)}$ and a control bit $t^{(b)}$. They generate the labels of children with the help of the corresponding correction word cw. As observed in [3], using the seed $\tilde{s}^{(b)}$ directly in the generation of both children and layer output would compromise its pseudo-randomness, which is required for security. Therefore, an extra PRG evaluation is employed to expand $\tilde{s}^{(b)}$ to a new pseudo-random seed $s^{(b)}$ for children and an element $w^{(b)}$ for the layer output.

Defending against a malicious dealer. The VDPF technique in [10] can ensure the dealer shares a well-formed point function. However, a malicious dealer may also attempt to cause a mismatch between the layer outputs and the point function outputs. For instance, the dealer secret-shares a point function $f_{\alpha,\beta}$ to two evaluators, while the layer outputs embedded in the function shares correspond to the binary representation of a different integer $\alpha' \neq \alpha$. To defend against this attack, we enable evaluators to verify that two trees defined by a pair of IVDPF keys only differ at one node per level. It forces that the nodes with different labels exactly form the special path. More specifically, we follow the verification method of the standard VDPF scheme in [10] to check each level. During the key generation, for each level i, we introduce a verification correction seed cs_i to correct the hash values of the different labels (i.e., the seeds and control bits) to be identical. In the verifiable evaluation, we let the evaluators check that all the corrected hash values are equal. If a correction seed can only correct at most one difference, this equality check guarantees that all other nodes at level i have the same labels on two trees.

Security. We show the security of our IVDPF in Fig. 4 with the following theorem, and its proof can be found in Appendix E.1.

Construction IVDPF

- **Parameters**: Let $\mathbb{G}, \mathbb{G}'_1, \ldots, \mathbb{G}'_n$ be Abelian groups. Let $\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{G}\}$ be a function family of point functions. Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$, $G'_0 : \{0,1\}^\lambda \to \mathbb{G}$, and $G'_i : \{0,1\}^\lambda \to \{0,1\}^\lambda \times \mathbb{G}'_i$ for $i \in [n]$ be pseudorandom generators. Let $H : \{0,1\}^{\lambda+1+n} \to \{0,1\}^{4\lambda}$ be a hash function sampled from a family $\mathcal{H}$ that is collision-resistant and XOR-collision-resistant, and $H' : \{0,1\}^{4\lambda} \to \{0,1\}^{2\lambda}$ be a hash function sampled from a family $\mathcal{H}'$ that is collision-resistant.

- $\mathsf{Gen}(1^\lambda, f_{\alpha,\beta} \in \mathcal{F}, \{\beta'_i \in \mathbb{G}'_i\}_{i\in[n]})$

Let $\alpha_1, \ldots, \alpha_n$ be the bits of α;
Sample random $s_0^{(0)}, s_0^{(1)} \leftarrow \{0,1\}^\lambda$;
$t_0^{(0)} := 0$; $t_0^{(1)} := 1$;
for $i := 1$ to n:
  $s_L^{(b)} || s_R^{(b)} || t_L^{(b)} || t_R^{(b)} \leftarrow G(s_{i-1}^{(b)})$ for $b \in \{0,1\}$;
  if $\alpha_i = 0$: $\mathsf{keep} := L$; $\mathsf{lose} := R$;
  else: $\mathsf{keep} := R$; $\mathsf{lose} := L$;
  $t_L^{\mathsf{cw}} := t_L^{(0)} \oplus t_L^{(1)} \oplus \alpha_i \oplus 1$;
  $t_R^{\mathsf{cw}} := t_R^{(0)} \oplus t_R^{(1)} \oplus \alpha_i$;
  $s^{\mathsf{cw}} := s_{\mathsf{lose}}^{(0)} \oplus s_{\mathsf{lose}}^{(1)}$; $\mathsf{cw}_i := (s^{\mathsf{cw}}, t_L^{\mathsf{cw}}, t_R^{\mathsf{cw}})$;
  $\tilde{s}_i^{(b)} := s_{\mathsf{keep}}^{(b)} \oplus t_{i-1}^{(b)} \cdot s^{\mathsf{cw}}$ for $b \in \{0,1\}$;
  $t_i^{(b)} := t_{\mathsf{keep}}^{(b)} \oplus t_{i-1}^{(b)} \cdot t_{\mathsf{keep}}^{\mathsf{cw}}$ for $b \in \{0,1\}$;
  $\alpha_{[1,i]} := \alpha_1 || \ldots || \alpha_i$;
  $\mathsf{cs}_i := H(\tilde{s}_i^{(0)} || t_i^{(0)} || \alpha_{[1,i]}) \oplus H(\tilde{s}_i^{(1)} || t_i^{(1)} || \alpha_{[1,i]})$;
  $s_i^{(b)} || w_i^{(b)} \leftarrow G'_i(\tilde{s}_i^{(b)})$;
  $\mathsf{lcw}_i := (-1)^{t_i^{(1)}} \cdot [\beta'_i - w_i^{(0)} + w_i^{(1)}]$;
$\mathsf{ocw} := (-1)^{t_n^{(1)}} \cdot (\beta - G'_0(s_n^{(0)}) + G'_0(s_n^{(1)}))$;
$f_{\alpha,\beta}^{(0)} := (s_0^{(0)}, (\mathsf{cw}_i, \mathsf{cs}_i, \mathsf{lcw}_i)_{i\in[n]}, \mathsf{ocw})$;
$f_{\alpha,\beta}^{(1)} := (s_0^{(1)}, (\mathsf{cw}_i, \mathsf{cs}_i, \mathsf{lcw}_i)_{i\in[n]}, \mathsf{ocw})$;
return $(f_{\alpha,\beta}^{(0)}, f_{\alpha,\beta}^{(1)})$.

- $\mathsf{Eval}(b, f_{\alpha,\beta}^{(b)}, X \subseteq \{0,1\}^n)$:

Parse $f_{\alpha,\beta}^{(b)} := (s_0, t_0, (\mathsf{cw}_i, \mathsf{lcw}_i)_{i\in[n]}, \mathsf{ocw})$;
$t_0 := b$; $\tau_i := \mathsf{cs}_i$; $\tau^{(b)} := 0$;
for $i \in [n]$: $z_{i,0}^{(b)} := 0$; $z_{i,1}^{(b)} := 0$.
for $x \in X$:
  Let $x_1, \ldots, x_n$ be the bits of x;
  for $i := 1$ to n:
    $s_L || s_R || t_L || t_R \leftarrow G(s_{i-1})$;
    Parse $\mathsf{cw}_j := (s^{\mathsf{cw}}, t_L^{\mathsf{cw}}, t_R^{\mathsf{cw}})$;
    if $x_i = 0$: $\mathsf{keep} := L$;
    else: $\mathsf{keep} := R$;
    $\tilde{s}_i || t_i := (s_{\mathsf{keep}} || t_{\mathsf{keep}}) \oplus t_{i-1} \cdot (s^{\mathsf{cw}} || t_{\mathsf{keep}}^{\mathsf{cw}})$;
    $x_{[1,i]} := x_1 || \ldots || x_i$;
    $\tau_i := \tau_i \oplus H'(\tau_i \oplus H(\tilde{s}_i || t_i || x_{[1,i]}) \oplus t_i \cdot \mathsf{cs}_i)$;
    $s_i || w_i \leftarrow G'_i(\tilde{s}_i)$;
    $z_{i,x_i}^{(b)} := z_{i,x_i}^{(b)} + (-1)^b \cdot (w_i + t_i \cdot \mathsf{lcw}_i)$;
  $y_x^{(b)} := (-1)^b \cdot (G'_0(s_n) + t_n \cdot \mathsf{ocw})$;
  for $i \in [n]$: $\tau^{(b)} := \tau^{(b)} \oplus H'(\tau^{(b)} \oplus \tau_i)$;
return $(\{y_x^{(b)}\}_{x\in X}, \{z_{i,0}^{(b)}, z_{i,1}^{(b)}\}_{i\in[n]}, \tau^{(b)})$.

- $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$

return $\tau^{(0)} = \tau^{(1)}$.

Figure 4: The construction of incremental VDPF.

Theorem 1. *Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$, $G'_0 : \{0,1\}^\lambda \to \mathbb{G}$, and $G'_i : \{0,1\}^\lambda \to \{0,1\}^\lambda \times \mathbb{G}'_i$ for $i \in [n]$ be pseudorandom generators. Let $H : \{0,1\}^{\lambda+1+n} \to \{0,1\}^{4\lambda}$ be a collision resistant and XOR-collision resistant hash function, and $H' : \{0,1\}^{4\lambda} \to \{0,1\}^{2\lambda}$ be a collision-resistant hash function. The IVDPF scheme in Fig. 4 is a secure IVDPF as described in Definition 2.*

5.2 Our construction using incremental VDPF

We describe our CDPF scheme from incremental VDPF (IVDPF-CDPF) in Fig. 5. It is aimed to constraint a function family $\mathcal{F} := \{f : \{0,1\}^n \to \tilde{\mathbb{G}}\}$ of point functions, where $\tilde{\mathbb{G}}$ is an Abelian group. Let $N := 2^n$. Our IVDPF-CDPF supports N constraint configurations while evaluators store only $2n$ public keys $\{k_{i,0}, k_{i,1}\}_{i\in[n]}$. The intuition is that, for a function $f_{\alpha,\beta}$ shared in the IVDPF scheme, the evaluators jointly select the verification key $\mathsf{vk}_\alpha := \prod_{i\in[n]}(k_{i,0})^{z_{i,0}} \cdot (k_{i,1})^{z_{i,1}}$ using the layer outputs, and check if the dealer holds a secret key encoding the discrete logarithm of vk_α.

To defend against the attack on the linear composition of discrete logarithms, we utilize a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$. Let g be the generator of $\mathbb{G}$. We let $\mathsf{Constraint}$ generate random public keys $(k_{i,0} := e(g,g)^{r_{i,0}}, k_{i,1} := e(g,g)^{r_{i,1}})_{i\in[n]}$ to represent configurations. Next, SKGen can product randomized secret keys using the master secret key $\mathsf{msk} := \{r_{i,0}, r_{i,1}\}_{i\in[n]}$. For each verification key vk_α, we have $d := \sum_{i\in[n]} r_{i,\alpha_i}$ such that $e(g,g)^d = \mathsf{vk}_\alpha$. A secret key for vk_α is computed by $\mathsf{sk}_\alpha := (g^c, d/c)$ where $c \leftarrow \mathbb{Z}_p^*$. Note that, $e(g^c, g^{d/c}) = e(g,g)^d$.

To secretly share a point function $f_{\alpha,\beta}$, the dealer uses algorithm FGen to generate function shares and proof shares. Concretely, the dealer first computes function shares by IVDPF. Next, it re-randomizes

Construction IVDPF-CDPF

- **Parameters**: Let $\tilde{\mathbb{G}}$ be an Abelian group, $\mathcal{F} := \{f : \{0,1\}^n \to \tilde{\mathbb{G}}\}$ be a point function family, s.t. $N := 2^n$ is an integer in $\mathsf{poly}(\lambda)$. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$ be a bilinear map, where $\mathbb{G}, \mathbb{G}_t$ are multiplicative cyclic groups with the prime order p, and the generator g of $\mathbb{G}$. Let $H : \mathbb{G}_t \to \{0,1\}^{2\lambda}$ and $H' : \{0,1\}^{n \cdot \lceil \log p \rceil} \to \{0,1\}^{2\lambda}$ be hash functions sampled from $\mathcal{H}, \mathcal{H}'$ that are collision-resistant.
- $\mathsf{Constraint}(1^\lambda, \{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i \in \mathbb{Z}_N})$:
  1. **assert** $\mathcal{Q}_i = \{f_{i,v}\}_{v \in \tilde{\mathbb{G}}}$ and $\forall a \in \mathbb{A}_i, |a| = 1$ for $i \in \mathbb{Z}_N$;
  2. **for** $j \in [n]$:
  3. $\quad r_{j,0}, r_{j,1} \leftarrow \mathbb{Z}_p$;
  4. $\quad k_{j,0} := e(g,g)^{r_{j,0}}, k_{j,1} := e(g,g)^{r_{j,1}}$;
  5. $\mathsf{msk} := (\{r_{j,0}, r_{j,1}\}_{j \in [n]}, \{\mathbb{A}_i\}_{i \in \mathbb{Z}_N})$;
  6. **return** $(\Lambda := \{(k_{j,0}, k_{j,1})\}_{j \in [n]}, \mathsf{msk})$.
- $\mathsf{SKGen}(1^\lambda, \mathsf{id}, \mathsf{msk})$:
  1. Parse $\mathsf{msk} := (\{r_{j,0}, r_{j,1}\}_{j \in [n]}, \{\mathbb{A}_i\}_{i \in \mathbb{Z}_N})$;
  2. $\mathsf{sk} := \emptyset$;
  3. **for** $i \in \mathbb{Z}_N$:
  4. $\quad$ Let $i_1, \ldots, i_n$ be the bits of i;
  5. $\quad$ **if** $\{\mathsf{id}\} \in \mathbb{A}_i$:
  6. $\quad\quad d := \sum_{j \in [n]} r_{j,i_j}$;
  7. $\quad\quad c \leftarrow \mathbb{Z}_p^*$; $\mathsf{sk}_{i,0} := g^c$; $\mathsf{sk}_{i,1} := d/c$;
  8. $\quad\quad \mathsf{sk}_i := (\mathsf{sk}_{i,0}, \mathsf{sk}_{i,1})$; $\mathsf{sk} := \mathsf{sk} \cup \{\mathsf{sk}_i\}$;
  9. **return** sk.
- $\mathsf{FGen}(\Lambda, f_{\alpha,\beta} \in \mathcal{F}, \mathsf{sk})$:
  1. $(f^{(0)}_{\alpha,\beta}, f^{(1)}_{\alpha,\beta}) \leftarrow \mathsf{IVDPF.Gen}(1^\lambda, f_{\alpha,\beta}, \{1 \in \mathbb{Z}_p\}_n)$;
  2. Choose sk_α from sk;
  3. Parse $\mathsf{sk}_\alpha := (\mathsf{sk}_0, \mathsf{sk}_1)$;
  4. $s \leftarrow \mathbb{Z}_p^*$; $u := (\mathsf{sk}_0)^s$; $v := \mathsf{sk}_1 \cdot s^{-1}$;
  5. $(v^{(0)}, v^{(1)}) \leftarrow \mathsf{AddShare}_{\mathbb{Z}_p,2}(v)$;
  6. $\pi^{(0)} := (u, v^{(0)})$; $\pi^{(1)} := (u, v^{(1)})$;
  7. **return** $((f^{(0)}_{\alpha,\beta}, \pi^{(0)}), (f^{(1)}_{\alpha,\beta}, \pi^{(1)}))$.
- $\mathsf{VEval}(b, \Lambda, f^{(b)}_{\alpha,\beta}, \pi^{(b)}, X \subseteq \mathbb{Z}_N)$:
  1. Parse $\pi^{(b)} := (u, v^{(b)})$, $\Lambda := \{(k_{i,0}, k_{i,1})\}_{i \in [n]}$;
  2. $(\{y^{(b)}_i\}_{i \in \mathbb{Z}_N}, Z, \tau^{(b)}_0) \leftarrow \mathsf{IVDPF.Eval}(b, f^{(b)}_{\alpha,\beta}, \mathbb{Z}_N)$;
  3. Parse $Z := \{z^{(b)}_{i,0}, z^{(b)}_{i,1}\}_{i \in [n]}$; Set $\tau^{(b)}_1 := 0$;
  4. **for** $i \in [n]$: $\delta_i := b + (-1)^b \cdot (z^{(b)}_{i,0} + z^{(b)}_{i,1})$;
  5. $\tau^{(b)}_1 := H(\delta_1 || \ldots || \delta_n)$;
  6. $\mathsf{vk}^{(b)} := \prod_{i \in [n]} (k_{i,0})^{z^{(b)}_{i,0}} \cdot (k_{i,1})^{z^{(b)}_{i,1}}$;
  7. $v := v^{(b)}$;
  8. **if** $b = 0$: $\tau^{(b)}_2 := H(e(u, g^v)/\mathsf{vk}^{(b)})$;
  9. **else**: $\tau^{(b)}_2 := H(\mathsf{vk}^{(b)}/e(u, g^v))$;
  10. **return** $(\{y^{(b)}_x\}_{x \in X}, \tau^{(b)} := (\tau^{(b)}_0, \tau^{(b)}_1, \tau^{(b)}_2))$.
- $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$:
  1. **return** $\mathsf{IVDPF.Verify}(\tau^{(0)}_0, \tau^{(1)}_0)$ $\wedge \forall i \in [2], \tau^{(0)}_i = \tau^{(1)}_i$.

Figure 5: The construction of faster CDPF from IVDPF.

$\mathsf{sk}_\alpha := (g^c, d/c)$ to (u, v), such that $e(g^c, g^{d/c}) = e(u, g^v)$, and produces the proof shares consisting of u and the additive shares of v.

Upon receiving the function shares with proof shares, in addition to evaluating function shares, the evaluators run VEval to compute $\langle e(u, g^v)/\mathsf{vk}_\alpha \rangle$ and interpret these secret shares as tokens $\tau^{(0)}_2, \tau^{(1)}_2$. They exchange tokens to check if $e(u, g^v)/\mathsf{vk}_\alpha = 1$ in Verify to make a decision. To ensure the dealer knows a valid secret key if Verify outputs 1, the evaluators additionally check if they obtain a verification key $\mathsf{vk}_\alpha \in \Lambda$. They first verify the well-formedness of function shares using IVDPF tokens $\tau^{(0)}_0, \tau^{(1)}_0$, which restricts that at most one non-zero value in each $\{z_{i,0}, z_{i,1}\}$. Next, they check if the non-zero value is equal to 1. More specifically, for each layer $i \in [n]$, each evaluator P_b computes the subtractive share of the difference between the value 1 and the sum of layer outputs as $\delta_i := b + (-1)^b \cdot (z^{(b)}_{i,0} + z^{(b)}_{i,1})$. Then P_b hashes these subtractive shares to obtain the audit token $\tau^{(b)}_1$. Clearly, $\tau^{(0)}_1 = \tau^{(1)}_1$ holds iff $\forall i \in [n]$, $z^{(0)}_{i,\alpha_i} + z^{(1)}_{i,\alpha_i} = 1$.

Security. We show the security of our IVDPF-CDPF in Fig. 5 with the following theorem, and its proof can be found in Appendix E.2.

Theorem 2. *Assume the decisional BDH assumption holds in the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$. Let $H : \{0,1\}^* \to \{0,1\}^{2\lambda}$ and $H' : \{0,1\}^{n \cdot \lceil \log p \rceil} \to \{0,1\}^{2\lambda}$ be two collision-resistant hash functions. Let* IVDPF *be a secure incremental VDPF. The IVDPF-CDPF in Fig. 5 is a secure constrained DPF as described in Definition 1.*

Construction $\star$-CDPF

- **Parameters**: Let ℓ, t be integers in $\mathsf{poly}(\lambda)$. Let $\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{F}_{2^t}\}$, $\mathcal{F}' := \{f : \{0,1\}^{n+\log \ell} \to \mathbb{F}_{2^t}\}$ be point function families, s.t. $N := 2^n$ is an integer in $\mathsf{poly}(\lambda)$. Let $H : \{0,1\}^{Nt} \to \{0,1\}^{2\lambda}$ be a hash function sampled from a family $\mathcal{H}$ that is collision-resistant.

- The SKGen algorithm always outputs a valid secret key $\perp$.

- $\mathsf{Constraint}(1^\lambda, \{\mathcal{Q}_i, \mathbb{A}_i\}_{i\in\mathbb{Z}_N})$:
  1. **assert** $\mathcal{Q}_i$ is represented by $\{\mathsf{rs}_{i,j} \in \mathbb{F}_{2^t}\}_{j\in\mathbb{Z}_\ell}$
  2. $\Lambda_i := \{\mathsf{rs}_{i,j}\}_{j\in\mathbb{Z}_\ell}, \forall i \in \mathbb{Z}_N$;
  3. **return** $(\Lambda := \{\Lambda_i\}_{i\in\mathbb{Z}_N}, \mathsf{msk} := \emptyset)$.

- $\mathsf{FGen}(\Lambda, f_{\alpha,\beta} \in \mathcal{F}, \perp)$:
  1. $(f^{(0)}_{\alpha,\beta}, f^{(1)}_{\alpha,\beta}) \leftarrow \mathsf{VDPF.Gen}(1^\lambda, f_{\alpha,\beta} \in \mathcal{F})$;
  2. Select $\mathsf{rs}_{\alpha,\rho}$ from Λ, s.t., $\beta \wedge \mathsf{rs}_{\alpha,\rho} = 0$;
  3. $\eta := \alpha \cdot \ell + \rho$;
  4. $(f^{(0)}_{\eta,\beta}, f^{(1)}_{\eta,\beta}) \leftarrow \mathsf{VDPF.Gen}(1^\lambda, f_{\eta,\beta} \in \mathcal{F}')$;
  5. $\pi^{(0)} := f^{(0)}_{\eta,\beta}$; $\pi^{(1)} := f^{(1)}_{\eta,\beta}$;
  6. **return** $((f^{(0)}_{\alpha,\beta}, \pi^{(0)}), (f^{(1)}_{\alpha,\beta}, \pi^{(1)}))$.

- $\mathsf{VEval}(b, \Lambda, f^{(b)}_{\alpha,\beta}, \pi^{(b)}, X \subseteq \mathbb{Z}_N)$:
  1. $(\{y^{(b)}_i\}, \tau^{(b)}_0) \leftarrow \mathsf{VDPF.Eval}(b, f^{(b)}_{\alpha,\beta}, \mathbb{Z}_N)$;
  2. $(\{\widetilde{y}^{(b)}_i\}, \tau^{(b)}_1) \leftarrow \mathsf{VDPF.Eval}(b, \pi^{(b)}, \mathbb{Z}_{N\ell})$;
  3. $\tau^{(b)}_2 := 0$;
  4. **for** $i \in \mathbb{Z}_N$: $\delta_i := y^{(b)}_i \oplus (\bigoplus_{j\in\mathbb{Z}_\ell} \widetilde{y}^{(b)}_{i\cdot\ell+j})$;
  5. $\tau^{(b)}_2 := H(\delta_0 || \ldots || \delta_{N-1})$;
  6. $\tau^{(b)}_3 := H(\bigoplus_{i\in\mathbb{Z}_N, j\in\mathbb{Z}_\ell} \mathsf{rs}_{i,j} \wedge \widetilde{y}^{(b)}_{i\cdot\ell+j})$;
  7. **return** $(\{y^{(b)}_x\}_{x\in X}, \tau^{(b)} := \{\tau^{(b)}_i\}_{i\in[0,3]})$.

- $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$:
  1. Parse $\tau^{(b)} := \{\tau^{(b)}_i\}_{i\in[0,3]}$ for $b \in \{0,1\}$;
  2. **return** $\forall i \in \{0,1\}$, $\mathsf{VDPF.Verify}(\tau^{(0)}_i, \tau^{(1)}_i)$ $\wedge\ \forall i \in \{2,3\}$, $\tau^{(0)}_i = \tau^{(1)}_i$.

Figure 6: The construction of $\star$-CDPF.

6 CDPF with Wildcard-style Constraint

In this section, we propose a DPF scheme with wildcard-style constraints ($\star$-CDPF) to restrict the function output. We note that, the purpose of previous CDPF schemes is to check whether the dealer is authorized to share a point function with the special input; while $\star$-CDPF is concerned with whether the secret-shared function is authorized w.r.t its special output. We adopt the verifiable DPF with auxiliary output as we mentioned in Sec. 1. Similar to [26, 27], our CDPF scheme are able to generalize to complex FSS classes derived from DPFs. For general, we consider that each constraint configuration Λ_i includes ℓ entries.

In our $\star$-CDPF, each authorized set $\mathcal{Q}_i$ is represented by ℓ restraint strings $\{\mathsf{rs}_{i,j}\}_{j\in\mathbb{Z}_\ell}$, such that $\mathcal{Q}_i := \{f_{i,v} \in \mathcal{F} | \exists j \in \mathbb{Z}_\ell, v \wedge \mathsf{rs}_{i,j} = 0\}$. A point function with special input i matches $\mathsf{rs}_{i,j}$ if and only if all restricted bits of its special output are equal to 0. For clarity, we let $\perp$ be a "valid secret key" for any authorized set $\mathcal{Q}_i$ in our $\star$-CDPF scheme. Note that, our $\star$-CDPF scheme can be extended to fine-grained constraints by setting each restraint string according to a verification key for a non-trivial secret key.

Fig. 6 presents our $\star$-CDPF construction for the function family $\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{F}_{2^t}\}$. The method $\mathsf{Constraint}$ sets each constraint configurations to the corresponding restraint string $\mathsf{rs}_{i,j}$. In FGen, the dealer generates VDPF keys for $f_{\alpha,\beta} \in \mathcal{F}$ as function shares. Then, the dealer selects the restraint string $\mathsf{rs}_{\alpha,\rho}$ by the condition $\beta \wedge \mathsf{rs}_{\alpha,\rho} = 0$, and generates a pair of VDPF keys $(f^{(0)}_{\eta,\beta}, f^{(1)}_{\eta,\beta})$ as proof shares, where $\eta := \alpha\ell + \rho$. During the evaluation process, the evaluators first execute the algorithm VEval to evaluate function shares and proof shares in VDPF scheme. Then the evaluators generate audit tokens. Concretely, they jointly compute $[\![\mathsf{rs}_{\alpha,\rho} \wedge \beta]\!]$ by calculating the inner product between the restraint strings $\{\mathsf{rs}_{i,j}\}_{i\in\mathbb{Z}_N, j\in\mathbb{Z}_\ell}$ and the evaluation results of $f_{\eta,\beta}$. Later, each evaluator P_b computes token $\tau^{(b)}_3$ by hashing the share $(\mathsf{rs}_{\alpha,\rho} \wedge \beta)^{(b)}$. In Verify, the equality check of $\tau^{(0)}_3 = \tau^{(1)}_3$ guarantees $\mathsf{rs}_{\alpha,\rho} \wedge \beta = 0$.

Moreover, we introduce $\tau^{(b)}_0, \tau^{(b)}_1, \tau^{(b)}_2$ for each P_b to prevent the malicious dealer from sharing a mismatched point function $f_{\eta',\beta'}$. The first two tokens $\tau^{(b)}_0, \tau^{(b)}_1$ are VDPF tokens of $f^{(b)}_{\alpha,\beta}, f^{(b)}_{\eta,\beta}$, respectively. They attest the well-formedness of shared point functions. The token $\tau^{(b)}_2$ is computed by hashing the concatenation of the XOR shares of $f_{\alpha,\beta}(i) \oplus (\bigoplus_{j\in\mathbb{Z}_\ell} f_{\eta,\beta}(i\cdot\ell + j))$ for all $i \in \mathbb{Z}_N$. Due to the collision-resistant H, the equality $\tau^{(0)}_2 = \tau^{(1)}_2$ in Verify holds only if $\eta \in [\alpha\ell, \alpha\ell + \ell)$ and the spacial output of the shared point functions are equal.

Security. We show the security of our $\star$-CDPF in Fig. 6 with the following theorem, and its proof can be found in Appendix E.3.

Theorem 3. *Let $H : \{0,1\}^{Nt} \to \{0,1\}^{2\lambda}$ be a collision-resistant hash function. Let* VDPF *be a secure verifiable DPF scheme. The $\star$-CDPF in Fig. 6 is a secure DPF with constraints as described in Definition 1.*

6.1 Efficiency

In this section, we offer an optimization for CDPF, and discuss the composition of CDPFs for different policies.

DPF tensoring. To support the general $\ell \geq 1$, we introduce a separate VDPF $f_{\eta,(\beta,1)}$ to select the target item in Λ. It results in $O(\lambda \cdot (n + \log \ell))$ communication overhead from the dealer to the evaluators. As observed in prior work [26, 27], the selecting VDPF $f_{\eta,(\beta,1)}$ can reuse the "backbone" of the VDPF $f_{\alpha,\beta}$, which is the DPF tensoring described by Boyle *et al.* [7]. Loosely speaking, the VDPF evaluation result of $f_{\alpha,\beta}$ can be considered as the n-th level results of $f_{\eta,(\beta,1)}$. Therefore, the selecting VDPF only needs to represent a point function with domain size ℓ, and then the communication overhead can be reduce to $O(\lambda \cdot \log \ell)$.

Multi-policy constraint. We observe that, for the same function family $\mathcal{F}$, different CDPF schemes share a common "backbone" of selecting VDPFs. The difference between their selecting VDPFs is the special auxiliary outputs with respect to the different constraints. Due to the privacy of VDPF, all items in the VDPF evaluation result are pseudo-random and independent to each other. Therefore, multiple CDPF schemes for the same $\mathcal{F}$ can form a composite construction for multi-policy constraints using a single selecting VDPF. Here, the selecting point function is converted to a new function with sufficient auxiliary outputs for all CDPFs. The security of the composite construction depends on the security of VDPF and the original CDPF schemes.

7 Implementation and Benchmark

Our CDPF schemes are implemented in Go and C. We implement the underlying DPF followed by [7, 10] and employ the optimization of the FSS tensor technique in [7]. AES-128 is chosen for PRG in DPF, and we implement it by Intel's AES-NI. We instantiate the bilinear map e in the elliptic curve BLS12-381 [1], and adopt the efficient BLS12-381 implementation in [21]. In addition, we use SHA-256 as the hash function in our constructions.

Our benchmarks are executed on a server with Intel Xeon Silver 4214 CPU at 2.20GHz running Ubuntu 20.04.5 LTS; with 48 vCPUs and 128 GB Memory. Each experiment result is an average from 500 - 1000 evaluations. For the purpose of comparison, we also perform the same benchmarks for the DPF-PACL and VDPF-PACL from [26, 27]. We rerun the source code [25] provided by the original authors, where the DPF-PACL uses the group $\mathbb{G}$ as the P-256 elliptic curve group, and the VDPF-PACL uses the group $\mathbb{G} := \mathbb{Z}_p^*$ with a 3072-bit prime p as specified in RFC3526 [20]. In addition, we evaluate a CDPF schemes for multi-policy constraints: the SK-$\star$-CDPF composed of the VDPF-PACL and our $\star$-CDPF.

Parameters. Set $\lambda = 128$ and $n = 32$. In practical, such as mailbox system [12], constraints are applied to a registered set X, and the evaluators evaluate DPF on X rather than the full domain. Denote $N = |X|$ as the number of constraint configurations, and ℓ as the number of entries per configuration.

Optimizations. We reduce the communication cost between the evaluators by XOR-accumulating some audit tokens, which are just verified by an equality check in Verify.

7.1 Proof time

Table 1 compares the proof time of schemes. That is, the running time of Prove in the DPF-PACL and VDPF-PACL; and the running time of FGen except the function share generation in our schemes. We note that, using the FSS tensor optimization, the computational complexity of proof time is log-linear in ℓ. The DPF-PACL and our $\star$-CDPF schemes achieve an overwhelming performance advantage, because their

operations are minimal except for (V)DPF key generation. Our IVDPF-CDPF scheme requires $O(n)$ group operations for layer outputs in the IVDPF key generation, resulting in additional running time. For the SK-$\star$-CDPF scheme, multiple constraints share a common selecting point function to save a VDPF generation. Due to the succinct of the underlying DPF representation, this optimization appears insignificant in the proof generation.

Table 1: Performance of proof generation.

# of entries per Λ_i	1	2^5	2^{10}	2^{15}
DPF-PACL [26, 27]	0.3 μs	1.37 μs	2.74 μs	3.13 μs
VDPF-PACL [26, 27]	30.1 ms	29.3 ms	29.9 ms	29.3 ms
IVDPF-CDPF (Ours)	31.7 μs	41.6 μs	50.2 μs	63.8 μs
$\star$-CDPF (Ours)	0.6 μs	1.49 μs	2.65 μs	3.05 μs
SK-$\star$-CDPF	29.1 ms	30.2 ms	29.4 ms	29.3 ms

7.2 Communication overhead

There are two communication rounds in our CDPF schemes and PACLs: one is that the dealer distributes the function shares and proof shares to the evaluators; the other is that two evaluators exchange their audit tokens. Table 2 reports the concrete communication overhead of the two rounds. Let s_ℓ be the standard DPF key size for the point functions with domain size ℓ. The VDPF key size of [10] is $s_\ell + 64$ bytes, and our IVDPF key size is $s_\ell + 96 \log \ell$ bytes due to $\log \ell$ correction seeds and layer correction words.

Table 2: Proof size and audit token size in bytes, where s_ℓ is the standard DPF key size for the domain size ℓ.

	Proof Size	Token Size
DPF-PACL [26, 27]	$32 + s_\ell$	64
VDPF-PACL [26, 27]	$1952 + s_\ell + 64$	816
Fixed VDPF-PACL	$1952 + s_\ell + 64$	880
IVDPF-CDPF (Ours)	$128 + s_\ell + 96 \log \ell$	64
$\star$-CDPF (Ours)	$s_\ell + 64$	64
SK-$\star$-CDPF	$1952 + s_\ell + 64$	880

The proof size and token size of our $\star$-CDPF and IVDPF-CDPF schemes are smaller than those of VDPF-PACL. Compared to the sum of multiple CDPFs, the SK-$\star$-CDPF scheme saves one VDPF key in proof and save one hash value in audit token. In particular, we compare our fixed VDPF-PACL with the VDPF-PACL in [26, 27]. As shown in Table 2, our security improvement is almost free, in fact it only increases the token size by 64 bytes.

7.3 Verification cost

The evaluators execute VEval (or Audit in PACLs) and Verify sequentially for verification. Thus, we take the total run-time of the two algorithm as the verification cost. To demonstrate the advantage of our IVDPF-CDPF scheme, we plot the run-time of verification by steps in Fig. 7, where $\ell = 1$. It shows that, for both DPF-PACL and VDPF-PACL, the most costly step is selecting the target verification key. Because they use N verification keys to represent N constraint configurations, and require $O(N)$ exponentiation/multiplication operations in the key selection step. Our IVDPF-CDPF reduces the constraint representation to $2 \log N$ public keys, so that only $O(\log N)$ exponentiation operations are required. Thus, our IVDPF-CDPF scheme is significantly more efficient than PACLs when N is large. For instance, our IVDPF-CDPF improves the performance of DPF-PACL by a factor of 2.5$\times$ for 2^{15} constraint configurations, and it is 2$\times$ faster than VDPF-PACL.

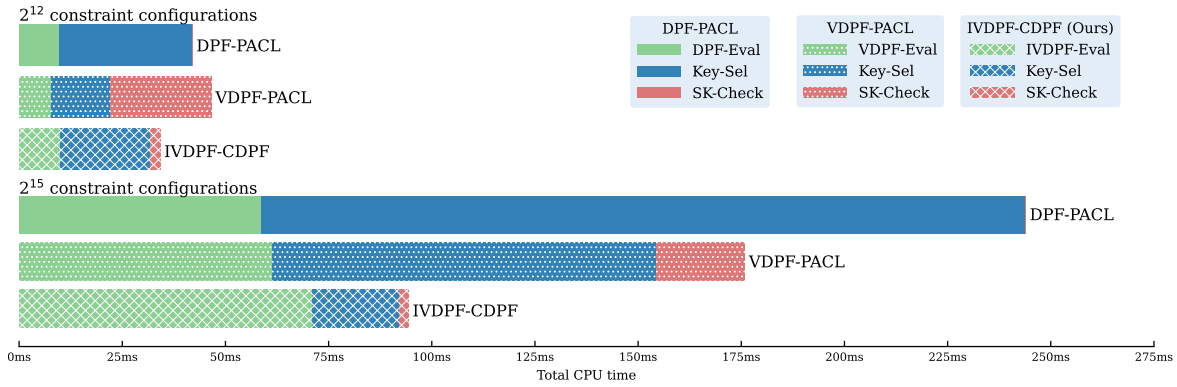

Figure 7: Verification performance of PACLs [26, 27] and our IVDPF-CDPF scheme.

In addition, we compare our $\star$-CDPF and the VDPF-PACL in Fig. 8, while the VDPF evaluation of $f_{\alpha,\beta}$ is the baseline. As the number of constraint configurations N increases, the amortized overhead of VDPF-PACL decreases. This is because the overhead of the SPoSS and the equality check are amortized over N. On the contrary, the amortized overhead of our $\star$-CDPF plateaus at the beginning due to its minimal operations. For the SK-$\star$-CDPF, its verification time is less than the sum of corresponding original schemes because it halves VDPF evaluations. This advantage would be significant when the input domain is large.

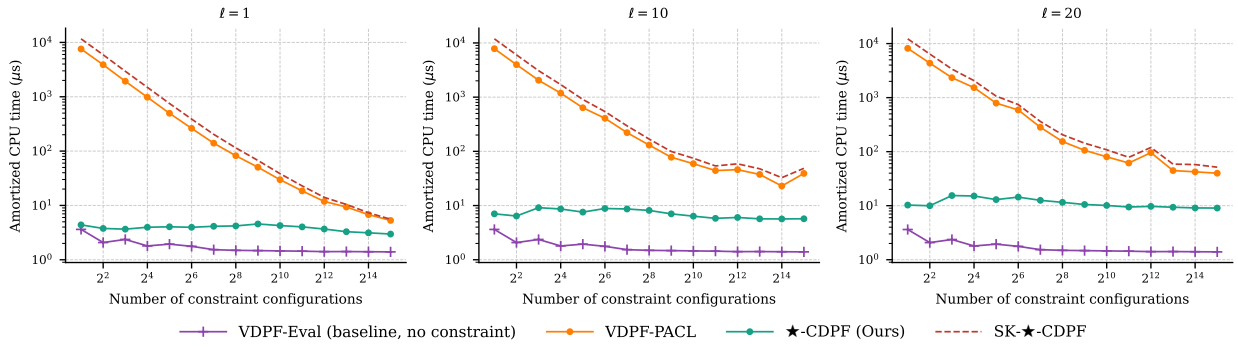

Figure 8: Comparison of the amortized verification cost, where VDPF-PACL refers to [26, 27]. The run-time is amortized by the number of constraint configurations N.

8 CDPF with Non-interactive Evaluators: Impossibility and Feasibility

To the best of our knowledge, all known CDPF constructions require the evaluators to have at least one round of communication with each other before the decision could be made. A natural question is whether there exists a CDPF scheme in which the evaluators do not need to communicate with each other. We call such type of CDPF schemes as "**n**on-**i**nteractive CDPF" and denote it as ni-CDPF. In this section, we provide the impossibility and feasibility results of ni-CDPF.

Impossibility. We show that it is impossible to construct a secure ni-CDPF in the plain model. Formally,

Theorem 4. *There exists no* ni-CDPF *construction in the plain model that is a secure CDPF as described in Definition 1.*

Proof. We now prove the theorem by contradiction. Assume there exists a secure CDPF with non-interactive evaluators in the plain model, denoted as $(\mathsf{Constraint}, \mathsf{SKGen}, \mathsf{FGen}, \mathsf{VEval}, \mathsf{Verify})$, for a function family $\mathcal{F} :=$

$\{f : \mathcal{D} \to \mathbb{G}\}$. That is, the verification result of P_b only depends on a single token $\tau^{(b)} \leftarrow \mathsf{Audit}(b, f^{(b)}, \pi^{(b)})$. Due to the privacy definition of CDPF, there is a PPT simulator Sim such that

$$\mathsf{View}_{\mathsf{CDPF}}(b, \Lambda, f, \mathsf{sk}, X) \approx_c \mathsf{Sim}(1^\lambda, b, \Lambda, X).$$

Now, we construct an efficient adversary $\mathcal{A}$ using Sim. On input $(1^\lambda, \Lambda)$, the adversary $\mathcal{A}$ proceeds as follows:

1 Pick a random subset X of $\mathcal{D}$;
2 $(f^{(0)}, \pi^{(0)}) \leftarrow \mathsf{Sim}(1^\lambda, 0, \Lambda, X)$;
3 $(f^{(1)}, \pi^{(1)}) \leftarrow \mathsf{Sim}(1^\lambda, 1, \Lambda, X)$;
4 Output $((f^{(0)}, f^{(1)}), (\pi^{(0)}, \pi^{(1)}), X)$.

According to the privacy definition, for any $b \in \{0, 1\}$, the output $(f^{(b)}, \pi^{(b)})$ of Sim passes the verification of P_b. Hence, $\mathcal{A}$ can win the soundness game with noticeable probability, which contradicts the security of the scheme. □

Feasibility. Our impossibility result above is based on the fact that if the evaluators do not communicate with each other, then they cannot coordinate with each other in the plain model. If the necessary "coordination" between the evaluators can be enabled, say through certain trusted setups, then we may be able to securely remove the "online" communication between the evaluators. If this is true, then we essentially obtain the first non-interactive CDPF!

Based on the above idea, we now describe a generic compiler for ni-CDPF using non-interactive zero-knowlege proofs (NIZKs)[6], PKI, and a broadcast channel. Let $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ denote the public-key encryption scheme. Each P_b generates a public-private key pair $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KGen}(1^\lambda)$, and then registers the public key pk_b to the PKI. The dealer obtains valid public keys $\mathsf{pk}_0, \mathsf{pk}_1$ from the PKI.

Given any secure CDPF scheme with one-round interactive evaluators, denoted as $\mathsf{CDPF} := (\mathsf{Constraint}, \mathsf{SKGen}, \mathsf{FGen}, \mathsf{VEval}, \mathsf{Verify})$, we can eliminate the interaction by letting the dealer broadcast the encrypted function shares, the corresponding tokens and an NIZK proof. More specifically, the dealer D first generates function shares with proof shares $((f^{(0)}, \pi^{(0)}), (f^{(1)}, \pi^{(1)}))$ by FGen following the CDPF. Next, for each $b \in \{0, 1\}$, D encrypts the function share $f^{(b)}$ with a randomness $r^{(b)}$ as $C^{(b)} := \mathsf{Enc}(\mathsf{pk}_b, (f^{(b)}, \pi^{(b)}), r^{(b)})$, then broadcasts the ciphertext $C^{(b)}$. For non-interactive verification, we let D compute and broadcast the token $\tau^{(b)}$ from $(Y^{(b)}, \tau^{(b)}) := \mathsf{VEval}(b, \Lambda, f^{(b)}, \pi^{(b)}, X)$ for each $b \in \{0, 1\}$, where Λ is the public constraint configuration list and X is the evaluation input set in the CDPF. If the dealer D provides correct tokens, each evaluator P_b can locally check if $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1$ to make a decision.

To prevent a malicious dealer D from forging $\tau^{(0)}, \tau^{(1)}$, we require the dealer D to provide an NIZK proof $\tilde{\pi}$ to prove the validity of tokens. Concretely, the statement consists of the ciphertexts and tokens being broadcast, the public keys, the configurations, and the input set, denoted as $\mathsf{st} := (C^{(0)}, \tau^{(0)}, C^{(1)}, \tau^{(1)}, \mathsf{pk}_0, \mathsf{pk}_1, \Lambda, X)$. The witness includes the function shares, the proof shares, and the randomness D uses in the encryption, denoted as $w := (f^{(0)}, \pi^{(0)}, r^{(0)}, f^{(1)}, \pi^{(1)}, r^{(1)})$. The NIZK proof $\tilde{\pi}$ is for the efficiently decidable binary relation

$$R := \left\{ (\mathsf{st}, w) \;\middle|\; \begin{array}{l} C^{(0)} = \mathsf{Enc}(\mathsf{pk}_0, (f^{(0)}, \pi^{(0)}), r^{(0)}) \;\wedge \\ C^{(1)} = \mathsf{Enc}(\mathsf{pk}_1, (f^{(1)}, \pi^{(1)}), r^{(1)}) \;\wedge \\ ((Y^{(0)}, t^{(0)}) := \mathsf{VEval}(0, \Lambda, f^{(0)}, \pi^{(0)}, X), \tau^{(0)} = t^{(0)}) \;\wedge \\ ((Y^{(1)}, t^{(1)}) := \mathsf{VEval}(1, \Lambda, f^{(1)}, \pi^{(1)}, X), \tau^{(1)} = t^{(1)}) \end{array} \right\}$$

Each P_b can locally verify the NIZK proof $\tilde{\pi}$ to check the validity of the tokens.

If the verification of $\tilde{\pi}$ passes and $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1$, P_b decrypts and accepts the function share $(f^{(b)}, \pi^{(0)}) := \mathsf{Dec}(\mathsf{sk}_b, C^{(b)})$, then evaluate the function share by executing $\mathsf{VEval}(b, \Lambda, f^{(b)}, \pi^{(b)}, X)$; otherwise, P_b rejects the function share.

We remark that, our purpose here is to illustrate the main ideas for constructing a non-interactive CDPF ni-CDPF scheme. When practical alternatives for NIZK are available, we indeed are able to obtain practical ni-CDPF schemes. In the Appendix D, we will propose a practical construction in the preprocessing model. Its verification is based on the *Vector Oblivious Linear Evaluation* (VOLE) correlations. As a subtle drawback of this scheme, it cannot ensure the honest evaluators reach consensus. In particular, a malicious dealer

[6]Note that, NIZK is usually realized in the random oracle and/or common reference string (CRS) models.

without valid secret key can successfully cheat one of the evaluators. Nevertheless, our practical ni-CDPF scheme can be securely used for private information retrieval applications [16, 29]. In these applications, the drawback of our scheme can be easily addressed by re-randomizing servers' (i.e., the CDPF evaluators) responses. Denote the original response of P_b as $\mathsf{res}^{(b)}$. Suppose servers agree on a random seed s in the initialization phase. If the server P_b accepts the function share, P_b responds to the user (i.e., the CDPF dealer) with re-randomized $\mathsf{res}^{(b)} + (-1)^b \cdot G(s)$ where G is a PRG, rather than the original value; otherwise, the server responds with $\perp$. Clearly, the user cannot learn any information of data from a single valid response.

References

[1] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 257–267, Amalfi, Italy, September 12–13, 2003. Springer, Heidelberg, Germany.

[2] Amos Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel, 1996.

[3] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy*, pages 762–776, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[4] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[5] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 871–900, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

[6] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[7] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303, Vienna, Austria, October 24–28, 2016. ACM Press.

[8] Paul Bunn, Jonathan Katz, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient 3-party distributed ORAM. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 215–232, Amalfi, Italy, September 14–16, 2020. Springer, Heidelberg, Germany.

[9] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.

[10] Leo de Castro and Antigoni Polychroniadou. Lightweight, maliciously secure verifiable function secret sharing. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 150–179, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany.

[11] Jack Doerner and abhi shelat. Scaling ORAM for secure computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 523–535, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[12] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 1775–1792. USENIX Association, August 11–13, 2021.

[13] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[14] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479, Singer Island, Florida, October 24–26, 1984. IEEE Computer Society Press.

[15] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. Available as Cryptology ePrint Archive Report 2006/309.

[16] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with popcorn. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 91–107, Santa Clara, CA, March 2016. USENIX Association.

[17] Keyu Ji, Bingsheng Zhang, Tianpei Lu, and Kui Ren. Multi-party private function evaluation for ram. *IEEE Transactions on Information Forensics and Security*, 18:1252–1267, 2023.

[18] Keyu Ji, Bingsheng Zhang, and Kui Ren. Fine-grained policy constraints for distributed point function. Cryptology ePrint Archive, Paper 2023/1672, 2023. https://eprint.iacr.org/2023/1672.

[19] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC, New York, 3rd edition, 2020. eBook Published on December 21, 2020.

[20] Mika Kojo and Tero Kivinen. More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). RFC 3526, 2003.

[21] Sam Kumar, Yuncong Hu, Michael P. Andersen, Raluca Ada Popa, and David E. Culler. JEDI: Many-to-many end-to-end encryption and key delegation for IoT. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019*, pages 1519–1536, Santa Clara, CA, USA, August 14–16, 2019. USENIX Association.

[22] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2505–2522. USENIX Association, August 12–14, 2020.

[23] Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. Spectrum: High-bandwidth anonymous broadcast. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 229–248, Renton, WA, April 2022. USENIX Association.

[24] Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis R. Bach. AriaNN: Low-interaction privacy-preserving deep learning via function secret sharing. *PoPETs*, 2022(1):291–316, January 2022.

[25] Sacha Servan-Schreiber, Simon Beyzerov, Eli Yablon, and Hyojae Park. PACL source code. github, 2022. https://github.com/sachaservan/pacl.

[26] Sacha Servan-Schreiber, Simon Beyzerov, Eli Yablon, and Hyojae Park. Private access control for function secret sharing. Cryptology ePrint Archive, Report 2022/1707, 2022. https://eprint.iacr.org/2022/1707.

[27] Sacha Servan-Schreiber, Simon Beyzerov, Eli Yablon, and Hyojae Park. Private access control for function secret sharing. In *2023 IEEE Symposium on Security and Privacy*, pages 809–828, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.

[28] Adithya Vadapalli, Kyle Storrier, and Ryan Henry. Sabre: Sender-anonymous messaging with fast audits. In *2022 IEEE Symposium on Security and Privacy*, pages 1953–1970, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press.

[29] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 299–313, Boston, MA, March 2017. USENIX Association.

[30] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22:265–279, 1981.

[31] Zhelei Zhou, Bingsheng Zhang, Hong-Sheng Zhou, and Kui Ren. Single-input functionality against a dishonest majority: Practical and round-optimal. Cryptology ePrint Archive, Paper 2024/305, 2024.

A Definition of VDPF

In this section, we recap the VDPF definition from [10].

Definition 3 (Verifiable Distributed Point Function [10])**.** *A 2-party VDPF scheme, parameterized by a function family $\mathcal{F}$ of point functions $f : \{0,1\}^n \to \mathbb{G}$, consists of three PPT algorithms* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Verify})$ *defined as follows:*

- $(f^{(0)}_{\alpha,\beta}, f^{(1)}_{\alpha,\beta}) \leftarrow \mathsf{Gen}(1^\lambda, f_{\alpha,\beta})$ *is the share generation algorithm that takes input as the security parameter 1^λ and a point function $f_{\alpha,\beta} \in \mathcal{F}$. It outputs a pair of VDPF keys, i.e., the additive shares of $[\![f_{\alpha,\beta}]\!]$.*
- $(\{y^{(b)}_x\}_{x\in X}, \tau^{(b)}) \leftarrow \mathsf{Eval}(b, f^{(b)}_{\alpha,\beta}, X)$ *is the verifiable evaluation algorithm that takes input as an index $b \in \{0,1\}$, a VDPF key $f^{(b)}_{\alpha,\beta}$, and a set of function inputs $X \subseteq \{0,1\}^n$. It outputs a tuple of values. The first set of values are the FSS outputs, which are additive shares of $f_{\alpha,\beta}(x), x \in X$. The second item is a token $\tau^{(b)}$ that is used to verify the well-formedness of the shared function.*
- 1 *or* $0 \leftarrow \mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$ *is the verification algorithm that takes input as a pair of tokens. It outputs* 1 *for acceptance or* 0 *for rejection.*

A secure VDPF must satisfy three properties as follows:

- ***Completeness.*** *For all $f \in \mathcal{F}$, $X \subseteq \{0,1\}^n$, it holds that*

$$\Pr\left[\begin{array}{l}(f^{(0)}, f^{(1)}) \leftarrow \mathsf{Gen}(1^\lambda, f);\\ (\{y^{(0)}_x\}, \tau^{(0)}) \leftarrow \mathsf{Eval}(0, f^{(0)}, X);\\ (\{y^{(1)}_x\}, \tau^{(1)}) \leftarrow \mathsf{Eval}(1, f^{(1)}, X) :\\ \forall x \in X,\ \ y^{(0)}_x + y^{(1)}_x = f(x)\ \ \wedge\\ \mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1\end{array}\right] = 1$$

- ***Privacy.*** *For a function $f \in \mathcal{F}$ and a set of inputs $X \subseteq \{0,1\}^n$, define the view of evaluator P_b $\mathsf{View}_{\mathsf{VDPF}}(b, f, X)$ as the probability distribution ensemble $\{(f^{(b)}, \tau^{(1-b)})\}_\lambda$, where $(f^{(0)}, f^{(1)}) \leftarrow \mathsf{Gen}(1^\lambda, f)$, $(\{y^{(1-b)}_x\}_{x\in X}, \tau^{(1-b)}) \leftarrow \mathsf{Eval}(1-b, f^{(1-b)}, X)$. There exists a PPT simulator Sim such that for all $f \in \mathcal{F}, X \subseteq \{0,1\}^n$, we have:*

$$\mathsf{View}_{\mathsf{VDPF}}(b, f, X) \approx_c \mathsf{Sim}(1^\lambda, b, n, \mathbb{G}, X)$$

- ***Soundness.*** *Let $f^{(b)}$ be the (possibly maliciously generated) share received by the evaluator P_b. There exists a negligible function negl such that for any $X \subseteq \{0,1\}^n$, $(\{y^{(b)}_x\}_{x\in X}, \tau^{(b)}) \leftarrow \mathsf{Eval}(b, f^{(b)}, X)$ for $b \in \{0,1\}$, it holds that*

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1 :\\ \left|\{x \in X | y^{(0)}_x + y^{(1)}_x \neq 0\}\right| \leq 1\end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

B Definition of PACL

For simplicity, we focus on 2-party FSS scheme, i.e. $(2,2)$-FSS, and then recap the PACL definition in [27]:

Definition 4. *Let integer $N := 2^n$ and $\mathcal{F} := \{f_i : \{0,1\}^n \to \{0,1\}^* | 1 \le i \le N\}$ be a family of functions. Let $(\mathsf{Gen}, \mathsf{Eval})$ instantiate a 2-party FSS scheme for $\mathcal{F}$. A PACL scheme consists of efficient algorithms $\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Audit}$ and Verify defined as follows:*

- $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$ *takes as input a security parameter 1^λ and a function $f \in \mathcal{F}$, and outputs a new pair of verification and access keys $(\mathsf{vk}, \mathsf{sk})$.*
- $(\pi^{(0)}, \pi^{(1)}) \leftarrow \mathsf{Prove}(f, \mathsf{sk})$ *takes as input a function $f \in \mathcal{F}$ and access key sk, and outputs a vector of 2 proof secret shares $(\pi^{(0)}, \pi^{(1)})$.*
- $\tau^{(b)} \leftarrow \mathsf{Audit}(b, \Lambda, f^{(b)}, \pi^{(b)})$ *takes as input access control list $\Lambda = (\mathsf{vk}_1, \ldots, \mathsf{vk}_N)$, function secret share $f^{(b)}$ of f sampled according to Gen, and proof share $\pi^{(b)}$, and outputs audit token $\tau^{(b)}$.*
- 1 *or* $0 \leftarrow \mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$ *takes as input a set of 2 audit tokens, and outputs 1 for acceptance or 0 for rejection.*

The above functionality must satisfy:

- ***Completeness.*** *A PACL scheme is complete if for all security parameters λ, for all $b \in \{0,1\}$, for all $\Lambda := (\mathsf{vk}_1, \ldots, \mathsf{vk}_N)$ where $\forall i, \mathsf{vk}_i$ is sampled according to KeyGen, and for all secret shares $(f^{(0)}, f^{(1)})$ of $f \in \mathcal{F}$ sampled according to $\mathsf{Gen}(1^\lambda, f)$, it holds that*

$$\Pr\left[\begin{array}{l} (\pi^{(0)}, \pi^{(1)}) \leftarrow \mathsf{Prove}(f, \mathsf{sk}); \\ (f^{(0)}, f^{(1)}) \leftarrow \mathsf{Gen}(1^\lambda, f); \\ \{\tau^{(b)} \leftarrow \mathsf{Audit}(b, \Lambda, f^{(b)}, \pi^{(b)}) | b \in \{0,1\}\} : \\ \mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = \mathsf{Check}(\Lambda, f, \mathsf{sk}) \end{array}\right] = 1$$

- ***Efficiency.*** *The size of each proof share $\pi^{(b)}$ is most $O(\lambda N^\epsilon)$ for any $\epsilon < 1$ (possibly dependent on n). The size of each audit token $\tau^{(b)}$ is $O(\lambda)$.*
- ***Privacy.*** *For any $b \in \{0,1\}$, define View_b to be the distribution over $\{(\pi^{(b)}, \tau^{*(b)})\} \cup \{\tau^{(1-b)}\}$ where $\pi^{(b)}$ is sampled according to $\mathsf{Prove}(f, \mathsf{sk})$, $\tau^{*(b)}$ is sampled arbitrarily and $\tau^{(1-b)}$ is generated by $\mathsf{Audit}(\Lambda, f^{(1-b)}, \pi^{(1-b)})$. A PACL is private if there exists an efficient simulator Sim such that*

$$\mathsf{View}_b \approx_c \mathsf{Sim}(1^\lambda, b, \Lambda, \tau^{*(b)}).$$

  That is, the distribution of proof shares and audit shares reveal nothing about f_i or the access key sk to any one computationally bounded (possibly malicious) verifier.
- ***Soundness.*** *For any PPT adversary $\mathcal{A}$ with oracle access to GetKey, it holds that*

$$\Pr[\mathbf{G}^{\mathsf{sound}}_{\mathsf{PACL},\mathcal{A}}(\lambda) = 1] \le \mathsf{negl}(\lambda)$$

  where $\mathbf{G}^{\mathsf{sound}}_{\mathsf{PACL},\mathcal{A}}(\lambda)$ is depicted in Fig. 9.

First, notice that the scheme allows simultaneous messaging, i.e., all the parties can send messages to each other in the same round. Following the simulation paradigm, typically, rushing adversary model is considered; that is, the adversary's messages can be generated at the end of the round, after seeing the honest parties' messages. However, in the privacy definition of [27], the simulator is given the adversary's message as input, which deviates from the reality.

Second, during the real execution, each evaluator receives a function share and a proof share from the dealer as well as an audit token from the other evaluator. However, in the privacy definition of [27], the view of evaluator does not include the function share. We show a counterexample to the claim that this privacy definition is problematic. Given any secure $\mathsf{PACL} := (\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Audit}, \mathsf{Verify})$ of a 2-party VDPF for a point function family $\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{G}\}$ under their definition, we construct a new audit algorithm $\mathsf{Audit}'(\Lambda, f^{(b)}, \pi^{(b)})$ that outputs $\tau'^{(b)} := (\tau^{(b)} \leftarrow \mathsf{Audit}(\Lambda, f^{(b)}, \pi^{(b)}), f^{(b)})$, and then obtain a new PACL scheme $\mathsf{PACL}' := (\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Audit}', \mathsf{Verify})$. Easy to note that, PACL' leaks the secret f. Unfortunately, we show that it is a secure PACL under their definition. Obviously, PACL' satisfies the completeness and soundness property due to the security of PACL. For the privacy of this PACL, the view

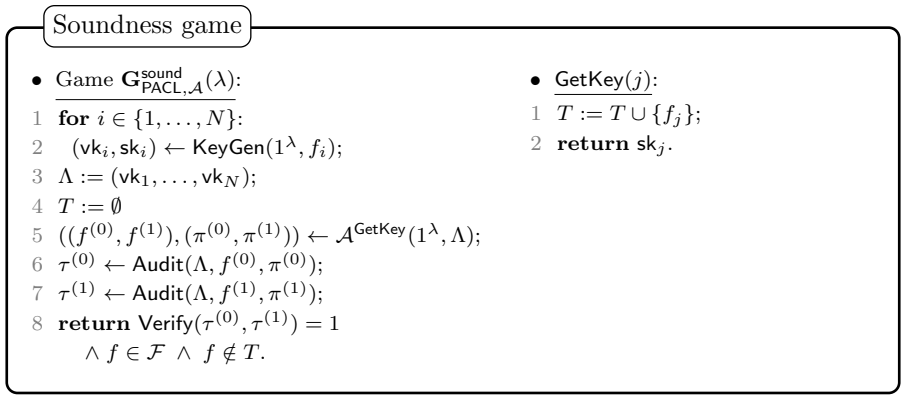

Figure 9: Soundness game for the PACL scheme $\mathsf{PACL} = (\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Audit}, \mathsf{Verify})$ for the function family $\mathcal{F}$.

of P_b is $\{(\pi^{(b)}, \tau^{*(b)})\} \cup \{\tau'^{(1-b)}\}$ where $\tau'^{(1-b)} \leftarrow \mathsf{Audit}'(\Lambda, f^{(1-b)}, \pi^{(1-b)})$. Denote the efficient simulator of PACL as $\mathsf{Sim}_{\mathsf{PACL}}$, the efficient simulator of VDPF as $\mathsf{Sim}_{\mathsf{VDPF}}$. We construct an efficient simulator Sim' for PACL' as follows: On input $(1^\lambda, b, \Lambda, \tau^{*(b)})$, Sim' proceeds

1 $(\pi^{(b)}, \tau^{*(b)}, \tau^{(1-b)}) \leftarrow \mathsf{Sim}_{\mathsf{PACL}}(1^\lambda, b, \Lambda, \tau^{*(b)})$;
2 $(f^{(1-b)}, -) \leftarrow \mathsf{Sim}_{\mathsf{VDPF}}(1^\lambda, 1-b, \mathcal{F}, \mathbb{Z}_N)$;
3 Output $(\pi^{(b)}, \tau^{*(b)}, (\tau^{(1-b)}, f^{(1-b)}))$.

Due to the privacy of PACL and VDPF, the output distribution of Sim is computationally indistinguishable from the distribution of real view of P_b. Therefore, PACL' indeed leaks the function f but is a secure scheme under the definition of [27]. In addition, to satisfy the efficiency property that the audit token size is $O(\lambda)$, we can make Audit' replace $f^{(b)}$ in the output with $y_1^{(b)} \in \mathbb{G}$ generated by $\mathsf{VDPF.Eval}(b, f^{(b)}, \{1\})$, which still leaks some information of the secret f.

C Properties of Definiton 1 for the Special Case

If the verification result of P_b only depends on $\tau^{(b)}$, the security properties of Definiton 1 are defined as follows:

- **Completeness.** For any $\{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N}$ such that $\mathsf{Constraint}(1^\lambda, \{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N}) \to (\Lambda, \mathsf{msk}) \neq \perp$, any $i \in \mathbb{Z}_N$, any function $f \in \mathcal{Q}_i$, any secret key $\mathsf{sk} \leftarrow \mathsf{SKGen}(1^\lambda, \mathsf{id}, \mathsf{msk})$ where id belongs to $\mathbb{A}_i$, and any $X \subseteq \mathcal{D}$, it holds that

$$\Pr\left[\begin{array}{l} ((f^{(0)}, \pi^{(0)}), (f^{(1)}, \pi^{(1)})) \leftarrow \mathsf{FGen}(\Lambda, f, \mathsf{sk}); \\ (\{y_x^{(0)}\}_{x\in X}, \tau^{(0)}) \leftarrow \mathsf{VEval}(0, \Lambda, f^{(0)}, \pi^{(0)}, X); \\ (\{y_x^{(1)}\}_{x\in X}, \tau^{(1)}) \leftarrow \mathsf{VEval}(1, \Lambda, f^{(1)}, \pi^{(1)}, X) : \\ \forall x \in X,\ y_x^{(0)} + y_x^{(1)} = f(x) \wedge \\ \mathsf{Verify}(\tau^{(0)}, \perp) = 1\ \wedge\ \mathsf{Verify}(\tau^{(1)}, \perp) = 1 \end{array}\right] = 1$$

- **Privacy.** For a constraint configuration list Λ, a function $f \in \mathcal{F}$, a secret key sk and a set of function inputs $X \subseteq \mathcal{D}$, define the distribution representing the view of P_b as

$$\mathsf{View}_{\mathsf{CDPF}}(b, \Lambda, f, \mathsf{sk}, X) := \{(f^{(b)}, \pi^{(b)})\}$$

where $((f^{(0)}, \pi^{(0)}), (f^{(1)}, \pi^{(1)})) \leftarrow \mathsf{FGen}(\Lambda, f, \mathsf{sk})$.
There exists a PPT simulator Sim such that for any Λ, msk generated from $\mathsf{Constraint}(1^\lambda, \{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N})$ for any $\{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N}$, any function f from any $\mathcal{Q}_i$, any secret key $\mathsf{sk} \leftarrow \mathsf{SKGen}(1^\lambda, \mathsf{id}, \mathsf{msk})$ where id belongs to $\mathbb{A}_i$, any $X \subseteq \mathcal{D}$, and any $b \in \{0,1\}$, the following two distributions are computationally indistinguishable:

$$\mathsf{View}_{\mathsf{CDPF}}(b, \Lambda, f, \mathsf{sk}, X) \approx_c \mathsf{Sim}(1^\lambda, b, \Lambda, X)$$

- **Soundness.** For any PPT adversary $\mathcal{A}$, it holds that

$$\Pr[\mathbf{G}^{\mathsf{sound}}_{\mathsf{CDPF},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where the game $\mathbf{G}^{\mathsf{sound}}_{\mathsf{CDPF},\mathcal{A}}(\lambda)$ is depicted in Fig. 10.

Soundness game

- Game $\mathbf{G}^{\mathsf{sound}}_{\mathsf{CDPF},\mathcal{A}}(\lambda)$:

$\{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N} \leftarrow \mathcal{A}(1^\lambda)$;
$w \leftarrow \mathsf{Constraint}(1^\lambda, \{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N})$;
if $w = \perp$: **return** 0; **else**: parse w as (Λ, msk);
$T := \emptyset$;
$((f^{(0)}, \pi^{(0)}), (f^{(1)}, \pi^{(1)}), X) \leftarrow \mathcal{A}^{\mathsf{GetKey}(\cdot)}(\Lambda)$;
for $b \in \{0, 1\}$: $(\{y_x^{(b)}\}_{x\in X}, \tau^{(b)}) \leftarrow \mathsf{VEval}(b, \Lambda, f^{(b)}, \pi^{(b)}, X)$;
if $\mathsf{Predicate}(f, X) = 1$: **return** 0;
return $(\mathsf{Verify}(\tau^{(0)}, \perp) = 1 \vee \mathsf{Verify}(\tau^{(1)}, \perp) = 1) \wedge (f \notin T)$.

- $\mathsf{GetKey}(\mathsf{id})$:

$\mathsf{sk} \leftarrow \mathsf{SKGen}(1^\lambda, \mathsf{id}, \mathsf{msk})$;
for $i \in \mathbb{Z}_N$: **if** $\{\mathsf{id}\} \in \mathbb{A}_i$, $T := T \cup \mathcal{Q}_i$;
return sk.

- $\mathsf{Predicate}(f, X)$:

if $f \notin \mathcal{F}$: **return** 0;
$((f^{(0)}, f^{(1)}), (\pi^{(0)}, \pi^{(1)})) \leftarrow \mathsf{FGen}(\Lambda, f, \perp)$;
for $b \in \{0, 1\}$: $(\{y_x^{(b)}\}_{x\in X}, \tau^{(b)}) \leftarrow \mathsf{VEval}(b, \Lambda, f^{(b)}, \pi^{(b)}, X)$;
return $(\mathsf{Verify}(\tau^{(0)}, \perp) = 1) \ \wedge \ (\mathsf{Verify}(\tau^{(1)}, \perp) = 1)$.

Figure 10: Soundness game for constrained distributed point function $\mathsf{CDPF} = (\mathsf{Constraint}, \mathsf{SKGen}, \mathsf{FGen}, \mathsf{VEval}, \mathsf{Verify})$ for function family $\mathcal{F}$.

D CDPF with Non-Interactive Evaluators in the Preprocessing Model

In this section, we propose a practical ni-CDPF scheme, i.e., CDPF with non-interactive evaluators, in the preprocessing model. We assume all parties are connected by pairwise secure channels and a broadcast channel.

For readability, we present our ni-CDPF scheme as a protocol $\Pi_{\mathsf{ni\text{-}CDPF}}$ in Fig. 11. It is based on the VDPF technique from [10] and multi-verifier (subfield) VOLE from [31]. Similar to the VDPF-PACL in [26,27], our constraint configuration list $\Lambda := \{\Lambda_i\}$ is a series of group elements $\Lambda_i \in \mathbb{G} := \mathbb{Z}_p^*$, named verification keys. Let g be the generator of $\mathbb{G}$. The discrete logarithm of each verification key Λ_i to the base g is the secret key of Λ_i. Therefore, given a shared point function with special input α, the evaluators check if the dealer knows the correct discrete logarithm of Λ_α to make a decision.

As mentioned in Sec. 1, we first need to ensure that the dealer D distributes a pair of well-formed VDPF keys for a point function $f_{\alpha,\beta}$ with an auxiliary output 1. Then the evaluators can obliviously obtain the shared valid verification key $\langle\Lambda_\alpha\rangle$ over $\mathbb{Z}_p$. Naively, the verification of VDPF keys requires one-round interaction between the evaluators [10]. Indeed, we observe that, checks for well-formedness and the auxiliary output is independent of the real point function $f_{\alpha,\beta}$. Thus, the dealer can generate and distribute a pair of VDPF keys $K^{(0)}, K^{(1)}$ for a random special input α', without the output correction word for the special output β in the preprocessing phase; while the evaluators jointly check the valid of $K^{(0)}, K^{(1)}$. In this online phase, the dealer can broadcasts $\delta_\alpha := \alpha - \alpha'$ and the output correction word cw for β to make VDPF keys complete.

Protocol $\Pi_{\mathsf{ni\text{-}CDPF}}$

Parameters: A prime number p, a group $\mathbb{G} := \mathbb{Z}_p^*$ with a generator g, a family $\mathcal{F} := \{f : \{0,1\}^n \to \tilde{\mathbb{G}}\}$ of point functions, an integer $N := 2^n$, a hash function $H : \mathbb{Z}_p \to \{0,1\}^\lambda$ sampled from a collision resistant family.

Inputs: The dealer D and evaluators $\mathsf{P}_0, \mathsf{P}_1$ hold constraint configuration list $\Lambda \in \mathbb{G}^N$. The dealer D additionally holds the point function $f_{\alpha,\beta}$ and secret key sk.

Preprocessing Phase: The point function $f_{\alpha,\beta}$ and secret key sk are unknown by D.

1. D generates and distributes incomplete VDPF keys, and $\mathsf{P}_0, \mathsf{P}_1$ check if these keys are valid.
   (a) D picks random $\alpha' \leftarrow \mathbb{Z}_N$, generates a pair of VDPF keys for the point function $f^{(0)}_{\alpha',(\beta,1)}$, except for the correction word for the unknown β. Denote these incomplete VDPF keys as $K^{(0)}, K^{(1)}$. D distributes $K^{(0)}, K^{(1)}$ to $\mathsf{P}_0, \mathsf{P}_1$.
   (b) Each P_b evaluates $K^{(b)}$ on the input set $\mathbb{Z}_N$, except for the first output item correction, to obtain the VDPF audit token $\tau^{(b)}$ and the secret share $\{y_i^{(b)}\}_{i\in\mathbb{Z}_N}$ of the second output item, and computes $\delta^{(b)} := H(b + (-1)^b \cdot \sum_{i\in\mathbb{Z}_N} y_i^{(b)})$.
   (c) P_0 and P_1 exchange $\tau^{(0)}, \tau^{(1)}$ and $\delta^{(0)}, \delta^{(1)}$, and check if $\mathsf{VDPF.Verify}(\tau^{(0)}, \tau^{(1)}) = 1$ and $\delta^{(0)} = \delta^{(1)}$.
2. D picks random $v \leftarrow \mathbb{Z}_{p-1}$ and broadcasts $t := g^v \in \mathbb{G}$. Then, P_0 and P_1 agree on a random $c \in \mathbb{Z}_{p-1}$ and send c to D.
3. D and $\mathsf{P}_0, \mathsf{P}_1$ jointly generates VOLE correlations.
   (a) D and $\mathsf{P}_0, \mathsf{P}_1$ send $(\mathsf{init}, \mathsf{sid})$ to $\mathcal{F}^p_{\mathsf{mv\text{-}VOLE}}$. Then $\mathcal{F}^p_{\mathsf{mv\text{-}VOLE}}$ returns $\Delta^{(b)}$ to each P_b.
   (b) D picks random $w_0, w_1, w_2 \leftarrow \mathbb{Z}_p$ and sets $\boldsymbol{w} := (w_0, w_1, w_2)$;
   (c) D sends $(\mathsf{extend}, \mathsf{sid}, 3, \boldsymbol{w})$ to the functionality $\mathcal{F}^p_{\mathsf{mv\text{-}VOLE}}$ while $\mathsf{P}_0, \mathsf{P}_1$ sends $(\mathsf{extend}, \mathsf{sid}, 3)$ to $\mathcal{F}^p_{\mathsf{mv\text{-}VOLE}}$. Then $\mathcal{F}^p_{\mathsf{mv\text{-}VOLE}}$ returns $(\boldsymbol{m}^{(0)}, \boldsymbol{m}^{(1)})$ to D and returns $\boldsymbol{k}^{(b)}$ to each P_b, such that $\forall i \in \mathbb{Z}_3$, $k_i^{(b)} = m_i^{(b)} + w_i \cdot \Delta^{(b)}$.

Online Phase: D knows the point function $f_{\alpha,\beta}$ and secret key sk

1. D computes and broadcasts $\delta_\alpha := \alpha - \alpha' \in \mathbb{Z}_N$ and the correction word cw for β corresponding the incomplete keys $K^{(0)}, K^{(1)}$. All parties locally combine $\delta_\alpha, \mathsf{cw}$ and the incomplete keys to obtain the VDPF keys $f^{(0)}_{\alpha,(\beta,1)}$ and/or $f^{(1)}_{\alpha,(\beta,1)}$.
2. D and $\mathsf{P}_0, \mathsf{P}_1$ compute the correlations of the target verification key:
   (a) For each $b \in \{0,1\}$: D computes $(\{(\beta_i, y_i)\}_{i\in\mathbb{Z}_N}, \tau^{(b)}) := \mathsf{VDPF.Eval}(b, f^{(b)}_{\alpha,(\beta,1)}, \mathbb{Z}_N)$, $\mathsf{vk}^{(b)} := \prod_{i\in\mathbb{Z}_N} (\Lambda_i)^{y_i}$, $m^{(0)}_{\mathsf{vk},b} := m_b^{(0)}$, $m^{(1)}_{\mathsf{vk},b} := m_b^{(1)}$, and $w_{\mathsf{vk},b} := (\mathsf{vk}^{(b)})^c$. Then D broadcasts $\delta_b := w_{\mathsf{vk},b} - w_b$.
   (b) Each P_b locally computes $k^{(b)}_{\mathsf{vk},0} := k_0^{(b)} + \delta_0 \cdot \Delta^{(b)}$ and $k^{(b)}_{\mathsf{vk},1} := k_1^{(b)} + \delta_1 \cdot \Delta^{(b)}$.
3. P_0 and P_1 check whether these correlations are valid:
   (a) For each $b \in \{0,1\}$: D sends $m^{(b)}_{\mathsf{vk},b}$ to P_b.
   (b) Each P_b evaluates $(\{(\beta_i, y_i)\}_{i\in\mathbb{Z}_N}, \tau^{(b)}) := \mathsf{VDPF.Eval}(b, f^{(b)}_{\alpha,(\beta,1)}, \mathbb{Z}_N)$, computes $\mathsf{vk}^{(b)} := \prod_{i\in\mathbb{Z}_N} (\Lambda_i)^{y_i}$, and checks if $k^{(b)}_{\mathsf{vk},b} = m^{(b)}_{\mathsf{vk},b} + (\mathsf{vk}^{(b)})^c \cdot \Delta^{(b)}$.
4. P_0 and P_1 verify the secret key of D:
   (a) D computes $w_{\mathsf{vk}} := (\Lambda_\alpha)^c$, $m^{(0)}_{\mathsf{vk}} := m_2^{(0)}$ and $m^{(1)}_{\mathsf{vk}} := m_2^{(1)}$. D broadcasts $\delta_2 := w_{\mathsf{vk}} - w_2$ and $r := v - c \cdot \mathsf{sk}$ while for each $b \in \{0,1\}$: D sends $m_t^{(b)} := g^r \cdot m^{(b)}_{\mathsf{vk}}$ to P_b.
   (b) Each P_b computes $k^{(b)}_{\mathsf{vk}} := k_2^{(b)} + \delta_2 \cdot \Delta^{(b)}$, $k_t := g^r \cdot k^{(b)}_{\mathsf{vk}}$ and checks if $k_t^{(b)} = m_t^{(b)} + t \cdot \Delta^{(b)}$.
5. P_0 and P_1 verify the correlation of vk is computed correctly:
   (a) For each $b \in \{0,1\}$: D sends $A_0^{(b)} := m^{(b)}_{\mathsf{vk},0} \cdot m^{(b)}_{\mathsf{vk},1}$ and $A_1^{(b)} := m^{(b)}_{\mathsf{vk},0} \cdot w_{\mathsf{vk},1} + m^{(b)}_{\mathsf{vk},1} \cdot w_{\mathsf{vk},0} - m^{(b)}_{\mathsf{vk}}$ to P_b.
   (b) Each P_b locally computes $B^{(b)} := k^{(b)}_{\mathsf{vk},0} \cdot k^{(b)}_{\mathsf{vk},1} - k^{(b)}_{\mathsf{vk}} \cdot \Delta^{(b)}$ and checks if $B^{(b)} = A_0^{(b)} + A_1^{(b)} \cdot \Delta^{(b)}$.

Figure 11: Protocol for CDPF with non-interactive evaluators in the $\mathcal{F}^p_{\mathsf{mv\text{-}VOLE}}$-hybrid model

Next, we utilize Schnorr's protocol to prove that D knows the discrete logarithm sk of the select $\mathsf{vk} := \Lambda_\alpha$. Obviously, the commitment and challenge steps of Schnorr's protocol is independent of the statement vk and the witness sk and, thus they can be performed in the preprocessing phase. That is, D pre-broadcasts $t := g^v$ and then the evaluators $\mathsf{P}_0, \mathsf{P}_1$ sends a random $c \in \mathbb{Z}_{p-1}$ to D. The challenge is the contradiction between the privacy requirement that each P_b should not know the statement vk and the purpose that the evaluators can locally make a decision without communication to each other. Inspired by [31], we let the dealer commit to the statement (i.e., the secret shares of vk from the inner product between Λ and the evaluation results of VDPF keys) to the evaluators, then prove that the rest steps of Schnorr's protocol over vk and sk are processed in a proper way. In particular, these commitments are built by multi-verifier VOLE correlations, which is adopted from [31]. In Fig. 12, we recap the functionality of multi-verifier subfield VOLE of [31]

in our setting[7], and reduce it to a 2-verifier VOLE functionality $\mathcal{F}^p_{\mathsf{mv\text{-}VOLE}}$. According to Fig. 12, in the preprocessing phase, each P_b gets a global key $\Delta^{(b)}$ from the multi-verifier VOLE functionality $\mathcal{F}^p_{\mathsf{mv\text{-}VOLE}}$. For any vector $\boldsymbol{w} \in \mathbb{Z}_p^3$, D gets MAC tags $\boldsymbol{m}^{(0)}, \boldsymbol{m}^{(1)} \in \mathbb{Z}_p^3$ and the vector $\boldsymbol{w}$ from $\mathcal{F}^p_{\mathsf{mv\text{-}VOLE}}$; while each P_b gets the private key $\boldsymbol{m}^{(b)} \in \mathbb{Z}_p^3$, such that $\forall i \in \mathbb{Z}_3$, $k_i^{(b)} = m_i^{(b)} + w_i \cdot \Delta^{(b)}$.

In the online phase, the dealer first computes $\mathsf{vk}^{(0)}, \mathsf{vk}^{(1)}$ using the VDPF keys it distributes and commits to $(\mathsf{vk}^{(0)})^c$ and $(\mathsf{vk}^{(1)})^c$ by broadcasting $\delta_0 := (\mathsf{vk}^{(0)})^c - w_0$ and $\delta_1 := (\mathsf{vk}^{(1)})^c - w_1$ to all evaluators. Then the dealer commits to $\mathsf{vk}^c := (\mathsf{vk}^{(0)})^c \cdot (\mathsf{vk}^{(1)})^c$ by broadcasting $\delta_2 := \mathsf{vk}^c - w_2$. Each evaluator can verify these commitments after receiving the corresponding MAC tags from D privately. Finally, the dealer broadcasts $r := v - c \cdot \mathsf{sk}$ and the evaluators locally check if $g^r \cdot \mathsf{vk}^c = t$ using commitments to make a decision.

Security. We provide the intuition for the security of our $\Pi_{\mathsf{ni\text{-}CDPF}}$ in Fig. 11. First, the completeness clearly holds, except for the verification of vk^c (step 5 in the online phase). We note that, for each $b \in \{0,1\}$,

$$\begin{aligned} B^{(b)} &:= k^{(b)}_{\mathsf{vk},0} \cdot k^{(b)}_{\mathsf{vk},1} - k^{(b)}_{\mathsf{vk}} \cdot \Delta^{(b)} \\ &= (m^{(b)}_{\mathsf{vk},0} + (w_{\mathsf{vk},0} \cdot \Delta^{(b)}) \cdot (m^{(b)}_{\mathsf{vk},1} + w_{\mathsf{vk},1} \cdot \Delta^{(b)}) \\ &\quad - (m^{(b)}_{\mathsf{vk}} + w_{\mathsf{vk}} \cdot \Delta^{(b)}) \cdot \Delta^{(b)} \\ &= A_0^{(b)} + A_1^{(b)} \cdot \Delta^{(b)} + (w_{\mathsf{vk},0} \cdot w_{\mathsf{vk},1} - w_{vk}) \cdot (\Delta^{(b)})^2 \end{aligned}$$

where $w_{\mathsf{vk},0} := (\mathsf{vk}^{(0)})^c$, $w_{\mathsf{vk},1} := (\mathsf{vk}^{(1)})^c$ and $w_{\mathsf{vk}} := \mathsf{vk}^c$ if D is honest. Therefore, we have $B^{(b)} = A_0^{(b)} + A_1^{(b)} \cdot \Delta^{(b)}$.

For privacy, the preprocessing phase is independent of private input. In the online phase, the dealer broadcasts (1) δ_α, cw for VDPF keys, (2) $\delta_0, \delta_1, \delta_2$ for commitment, and (3) r for Schnorr's proof. It is easy to see, these values are uniformly distributed. In addition, the dealer sends (1) $m^{(b)}_{\mathsf{vk},b}$, (2) $m_t^{(b)}$ and (3) $A_0^{(b)}$, $A_1^{(b)}$ to P_b privately. The first two items open $w_{\mathsf{vk},b} := (\mathsf{vk}^{(b)})^c$ and $w_t := t$ to P_b, which are already known by P_b from the VDPF key and Schnorr's proof. The items in (3) are masked MAC tags and also reveal nothing of dealer's private input to P_b.

The soundness of our $\Pi_{\mathsf{ni\text{-}CDPF}}$ has a subtle drawback. Assume the discrete logarithm assumption holds in $\mathbb{G}$. A malicious dealer $\mathcal{A}$ can cheat one of the evaluators, say P_0, by broadcasting $\delta_1^* := (g^{\mathsf{sk}^*}/\mathsf{vk}^{(0)})^c - w_b$ for a forged secret key sk^*. Then P_0 would accept the forged proof from $\mathcal{A}$. Nevertheless, the other evaluator P_1 can distinguish the invalid δ_1^*. We conclude that any malicious dealer cannot cheat two evaluators at the same time unless it breaks the discrete logarithm assumption.

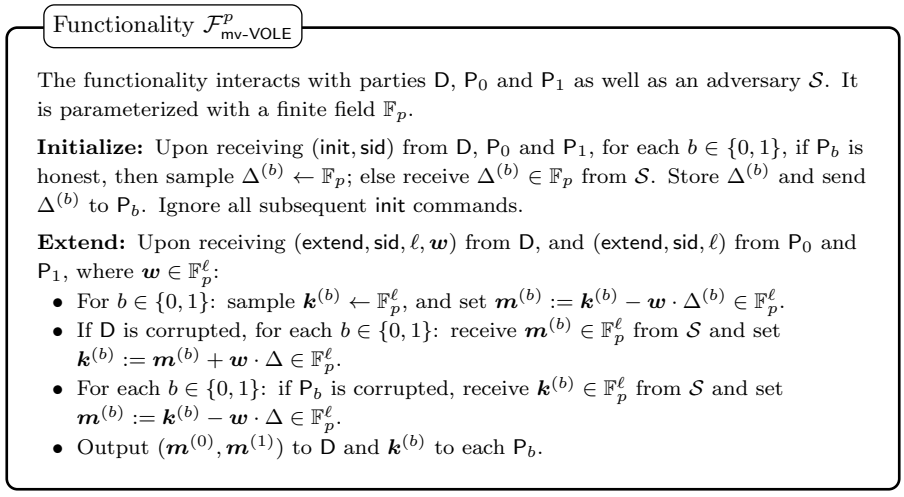

Figure 12: Functionality for multi-verifier VOLE [31]

[7]Recall that, we consider one dealer and two verifiers, and the malicious security for a single corrupted party.

E Security Proofs

E.1 Proof of Theorem 1

Proof. We prove the IVDPF construction in Fig. 4 satisfies the properties in Definition 2.

Completeness. Except for the layer outputs $\{z_{i,0}, z_{i,1}\}_{i\in[n]}$, our IVDPF is same as the incremental DPF construction of Boyle *et al.* [7]. Given the shared function $[\![f_{\alpha,\beta}]\!]$ and the input set X, for each level i, the layer outputs $z_{i,0}, z_{i,1}$ are generated by summing the i-bit prefix evaluation results of the point functions. Since only $\alpha_{[1,i]} := \alpha_1||\ldots||\alpha_i$ leads to a non-zero prefix evaluation result, we have $z^{(0)}_{i,\bar{\alpha}_i} + z^{(1)}_{i,\bar{\alpha}_i} = 0$ and $z^{(0)}_{i,\alpha_i} + z^{(1)}_{i,\alpha_i} = (\alpha_{[1,i]} \in X_{[1,i]}) \cdot \beta'_i$.

Privacy. We construct an efficient simulator Sim for the view of any evaluator P_b. On input $(1^\lambda, b, n, \mathbb{G}, \{\mathbb{G}'_i\}_{i\in[n]}, X)$, the simulator Sim proceeds as follows:

1 $s^{(b)}_0 \leftarrow \{0,1\}^\lambda$, $\mathsf{ocw} \leftarrow \mathbb{G}$
2 **for** $i := 1$ to n:
3 $\quad \mathsf{cw}_i \leftarrow \{0,1\}^{\lambda+2}$, $\mathsf{cs}_i \leftarrow \{0,1\}^{4\lambda}$, $\mathsf{lcw}_i \leftarrow \mathbb{G}'_i$
4 $f^* := (s^{(b)}_0, (\mathsf{cw}_i, \mathsf{cs}_i, \mathsf{lcw}_i)_{i\in[n]}, \mathsf{ocw})$
5 $(\{y^*_x\}, \{z^*_{i,0}, z^*_{i,1}\}, \tau^*) \leftarrow \mathsf{IVDPF.VEval}(b, f^*, X)$
6 Output (f^*, τ^*)

In the real view of P_b, the IVDPF key $f^{(b)}$ consists of (1) random seed $s^{(b)}_0$ which is distributed identically to the output of Sim, (2) the correction words $(\mathsf{cw}_i, \mathsf{cs}_i)_{i\in[n]}$, where each one is XOR'd with the output of the PRG G and (3) the output correction words $(\mathsf{lcw}_i)_{i\in[n]}, \mathsf{ocw}$, where each one is added with the pseudorandom output of G'_i. Since P_b does not known the seeds of G and G'_i, (2) and (3) are computationally indistinguishable from random elements. In addition, the audit token $\tau^{(1-b)}$ is equal to $\tau^{(b)}$, and the evaluator holding $f^{(b)}$ can locally compute it. Therefore, we conclude that the output of Sim is computationally indistinguishable from to the view of verifier P_b.

Soundness. Given the input set $X \subseteq \{0,1\}^n$, each evaluator has $(\{y^{(b)}_x\}_{x\in X}, \{z_{i,0}, z_{i,1}\}_{i\in[n]}, \tau^{(b)}) \leftarrow \mathsf{Eval}(b, f^{(b)}_{\alpha,\beta}, X)$. Denote $C_i := \{0,1\}/\{1 - j|z^{(0)}_{j,0} + z^{(1)}_{j,0} \neq 0, j \in \{0,1\}\}$. In addition, at every level i, each evaluator has a set of seeds and control bits, denoted as $\{\tilde{s}^{(0)}_{i,v}, \tilde{t}^{(0)}_{i,v},\}_{v\in X_{[1,i]}}$ and $\{\tilde{s}^{(1)}_{i,v}, t^{(1)}_{i,v}\}_{v\in X_{[1,i]}}$. The evaluators have the same correction seed cs_i. Let $\tilde{\tau}^{(b)}_{i,v} := H(\tilde{s}^{(b)}_{i,v}||t^{(b)}_{i,v}||v)$ and $\tau^{(b)}_{i,v} := \tilde{\tau}_{i,v} \oplus t^{(b)}_{i,v} \cdot \mathsf{cs}_i$ for $b \in \{0,1\}$. We consider two cases as follows:

Case 1: We prove that no PPT adversary $\mathcal{A}$ can construct IVDPF keys such that $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1$ when $\left|\{x \in X | y^{(0)}_x + y^{(1)}_x \neq 0\}\right| > 1$ or $\exists i \in [n], |C_i| < 1$. Suppose for contradiction that $\mathcal{A}$ constructs a key that satisfies the above condition. There exists $i \in [n]$, two distinct $u, v \in X_{[1,i]}$ such that $\tilde{s}^{(0)}_{i,u}||\tilde{t}^{(0)}_{i,u} \neq \tilde{s}^{(1)}_{i,u}||\tilde{t}^{(1)}_{i,u}$ and $\tilde{s}^{(0)}_{i,v}||\tilde{t}^{(0)}_{i,v} \neq \tilde{s}^{(1)}_{i,v}||\tilde{t}^{(1)}_{i,v}$, and $\forall x \in X_{[1,i]}$ we have $\tau^{(0)}_{i,x} = \tau^{(1)}_{i,x}$ to make $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1$ (due to the collision resistance of H'). Because of the collision resistance of H, we have $\tilde{\tau}^{(0)}_{i,u} \neq \tilde{\tau}^{(1)}_{i,u}$ and $\tilde{\tau}^{(0)}_{i,v} \neq \tilde{\tau}^{(1)}_{i,v}$. Hence, the adversary $\mathcal{A}$ needs to find cs_i such that $\mathsf{cs}_i = \tilde{\tau}^{(0)}_{i,u} \oplus \tilde{\tau}^{(1)}_{i,u} = \tilde{\tau}^{(0)}_{i,v} \oplus \tilde{\tau}^{(1)}_{i,v}$. It contradicts the XOR-collision resistance of H.

Case 2: We prove that no PPT adversary $\mathcal{A}$ can construct IVDPF keys such that $\exists x \in X, (\exists i \in [n], x_i \notin C_i) \wedge (y^{(0)}_x + y^{(1)}_x \neq 0)$, it holds that $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1$. Suppose for contradiction that there exists a level i such that for a node $u \in X_{[1,i]}$, such that $\tilde{s}^{(0)}_{i,u}||\tilde{t}^{(0)}_{i,u} = \tilde{s}^{(1)}_{i,u}||\tilde{t}^{(1)}_{i,u}$, but

$$G'_i(\tilde{s}^{(0)}_{i,u}) \neq G'_i(\tilde{s}^{(1)}_{i,u}).$$

Obviously, it contradicts the property of the PRG G'_i.

In conclusion, the construction in Fig. 4 satisfies the soundness property of Definition 2. □

E.2 Proof of Theorem 2

Proof. We prove the IVDPF-CDPF construction in Fig. 5 satisfies the properties in Definition 1.

Completeness. Due to the completeness of IVDPF, for the secret-shared point function $f_{\alpha,\beta}$, we have $[\![y_i]\!] = \beta$ if $i = \alpha$ and $[\![y_i]\!] = 0$ otherwise. As such,

$$\mathsf{vk} := \mathsf{vk}^{(0)} \cdot \mathsf{vk}^{(1)} = \prod_{i\in[n]} e(g,g)^{i,\alpha_i} = e(g,g)^{\sum_{i\in[n]} r_{i,\alpha_i}}$$

We show that for the secret key $\mathsf{sk} \leftarrow \mathsf{SKGen}(1^\lambda, \mathsf{id}, \mathsf{msk})$ where $\{\mathsf{id}\} \in \mathbb{A}_\alpha$, the algorithm Verify outputs 1. As described in Fig. 5, the algorithm Verify outputs 1 iff $\mathsf{IVDPF.Verify}(\tau_0^{(0)}, \tau_0^{(1)}) = 1$ and $\forall i \in [0,1], \tau_i^{(0)} = \tau_i^{(1)}$. The equality of the former follows from the completeness of IVDPF. The equality of $\tau_1^{(0)} = \tau_1^{(1)}$ follows from the special layer outputs equal to 1. To see why it holds that $\tau_2^{(0)} = \tau_2^{(1)}$, observe that $\mathsf{sk}_\alpha = \{\mathsf{sk}_0, \mathsf{sk}_1\} \in \mathsf{sk}$ and $e(u, g^v) = e(\mathsf{sk}_0, g^{\mathsf{sk}_1}) = \sum_{i\in[n]} r_{i,\alpha_i} = \mathsf{vk}$. Thus, it holds $\dfrac{e(u, g^{v^{(0)}})}{\mathsf{vk}^{(0)}} = \dfrac{\mathsf{vk}^{(1)}}{e(u, g^{v^{(0)}})}$, i.e., $\tau_2^{(0)} = \tau_2^{(1)}$, as required.

Privacy. We construct an efficient simulator Sim for the view of any evaluator P_b. We use the PPT simulator $\mathsf{Sim}_{\mathsf{IVDPF}}$ to generate the view of the IVDPF output. On input $(1^\lambda, b, \Lambda, X)$, the simulator Sim proceeds as follows:

1. $u \leftarrow \mathbb{G}_t$, $(v^{(0)}, v^{(1)}) \leftarrow \mathsf{AddShare}_{\mathbb{Z}_p,2}(0)$
2. $\pi^{(b)} := (u, v^{(b)})$
3. $(f^{(b)}, \tau_0^{(1-b)}) \leftarrow \mathsf{Sim}_{\mathsf{IVDPF}}(1^\lambda, b, \mathcal{F}, \mathbb{Z}_N)$
4. Compute $\tau_1^{(b)}, \tau_2^{(b)}$ using $f^{(b)}, \Lambda$ according to VEval
5. $\tau_1^{(1-b)} := \tau_1^{(b)}, \tau_2^{(1-b)} := \tau_2^{(b)}$
6. $\tau^{(1-b)} := (\tau_0^{(1-b)}, \tau_1^{(1-b)}, \tau_2^{(1-b)})$
7. Output $(f^{(b)}, \pi^{(b)}, \tau^{(1-b)})$

The output distribution of Sim is computationally indistinguishable from the distribution of the real view of P_b, because: (1) $v^{(b)}, u$ are distributed uniform in the real view; (2) function share $f^{(b)}$ and audit token $\tau_0^{(1-b)}$ are output by the simulator $\mathsf{Sim}_{\mathsf{VDPF}}$, which guarantees the computational indistinguishability; (3) audit tokens $\tau_1^{(1-b)}$, $\tau_2^{(1-b)}$ are equal to $\tau_1^{(b)}$, $\tau_2^{(b)}$ in the real view due to the CDPF completeness.

Soundness. Suppose for contradiction that there exists a PPT adversary $\mathcal{A}$ and non-negligible function δ such that:

$$\Pr[\mathbf{G}^{\mathsf{sound}}_{\mathsf{CDPF},\mathcal{A}}(\mathcal{A}) = 1] \geq \delta(\lambda)$$

Extract $f_{\alpha,\beta}$ from the output by $\mathcal{A}$ in $\mathbf{G}^{\mathsf{sound}}_{\mathsf{CDPF},\mathcal{A}}(\mathcal{A})$. By the soundness property of IVDPF, we can assume $f_{\alpha,\beta}$ are point functions with special input α and valid special layer outputs. Since H is a collision-resistant hash function, the inspection of $\tau_1^{(0)} = \tau_1^{(1)}$ restricts $\mathcal{A}$ to outputting IVDPF with special layer outputs equal to 1, which implies that both evaluators obtain secret shares of $\mathsf{vk} := e(g,g)^{\sum_{i\in[n]} r_{i,\alpha_i}}$.

We construct an efficient adversary $\mathcal{B}$ that breaks the decisional BDH assumption. On input $(1^\lambda, g^a, g^b, g^c, r)$, the adversary $\mathcal{B}$ proceeds as follows:

1. $\{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N} \leftarrow \mathcal{A}(1^\lambda, \mathcal{F})$
2. $(\Lambda, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \{(\mathcal{Q}_i, \mathbb{A}_i)\}_{i\in\mathbb{Z}_N})$
3. **assert** $\mathcal{Q}_i = \{f_{i,v}\}_{v\in\tilde{\mathbb{G}}}$ and $\forall a \in \mathbb{A}_i, |a| = 1$
4. $\alpha' \leftarrow \mathbb{Z}_N$;
5. Reset random k_{i,α'_i} in Λ, s.t., $\prod_{i\in[n]} k_{i,\alpha'_i} = e(g^a, g^b)$
6. Run $\mathcal{A}^{\mathsf{GetKey}}(1^\lambda, \Lambda)$ to get $f_{\alpha,\beta}, \pi := (u, v)$

7 **if** $\alpha' \neq \alpha$, repeat step 1 (up to λ times)

8 Compute $e(g,g)^{abc} := e(u, g^c)^v$ by the requirement $e(u, g^v) = e(g^a, g^b)$ for valid proof

9 Output $r = e(g,g)^{abc}$

Analyze the behavior of $\mathcal{B}$. The input to $\mathcal{B}$ is generated by uniform $a, b, c, z \in \mathbb{Z}_p$ and $r \leftarrow (e(g,g)^{abc}, e(g,g)^z)$. The adversary $\mathcal{B}$ runs $\mathcal{A}$ on the constraint configuration list Λ, which is constructed by the output of $\mathsf{Constraint}(1^\lambda, \mathcal{F})$ and $\mathsf{vk}_{i,\alpha'_i} = e(g^a, g^b)$. The view of $\mathcal{A}$ when run as a subroutine by $\mathcal{B}$ is distributed computationally indistinguishable to $\mathcal{A}$'s view in game $\mathbf{G}^{\mathsf{sound}}_{\mathsf{CDPF},\mathcal{A}}(\lambda)$, because $\{k_{i,\alpha'_i}\}_{i\in[n]}$ and $e(g^a, g^b)$ are distributed uniformly where a, b are uniformly random. Since $\mathcal{B}$ wins if the output of $\mathcal{A}$ is valid $f_{\alpha',\beta}, \pi$, and $\alpha' = \alpha$, we have that

$$
\begin{aligned}
&\left|\Pr[\mathcal{B}(e, g^a, g^b, g^c, r) = (r = e(g,g)^{abc})] - \frac{1}{2}\right| \\
&\geq (1 - (1 - \frac{1}{N})^\lambda) \cdot \Pr[\mathbf{G}^{\mathsf{sound}}_{\mathsf{CDPF},\mathcal{A}}(\lambda) = 1] \\
&\geq (1 - e^{-\frac{\lambda}{N}}) \cdot \delta(\lambda) \\
&\geq \begin{cases} \frac{\lambda}{2N} \cdot \delta(\lambda) & 0 \leq \frac{\lambda}{N} \leq 1.59 \\ (1 - e^{-1.59}) \cdot \delta(\lambda) & \frac{\lambda}{N} > 1.59 \end{cases}
\end{aligned}
$$

Since N is polynomial in λ, $\frac{\lambda}{2N}$ is non-negligible in λ. Thus $\mathcal{B}$ succeeds to distinguish the tuples $(g^a, g^b, g^c, e(g,g)^{abc})$ and $(g^a, g^b, g^c, e(g,g)^z)$ with a non-negligible advantage, contradicting the decisional BDH assumption. □

E.3 Proof of Theorem 3

Proof. We prove the $\star$-CDPF construction in Fig. 6 satisfies the properties in Definition 1.

Completeness. Due to the completeness of VDPF, for the secret-shared point function $f_{\alpha,\beta}$, we have $[\![y_i]\!] = \beta$ if $i = \alpha$ and $y_i = 0$ otherwise. For the secret-shared point function $f_{\eta,\beta}$, we have $[\![\tilde{y}_i]\!] = \beta$ if $j = \eta$ and $[\![\tilde{y}_i]\!] = (0,0)$ otherwise. We show that if there exists ρ such that $\beta \wedge \mathsf{rs}_{\alpha,\rho} = 0$ (i.e., $f_{\alpha,\beta} \in \mathcal{Q}_{\alpha,\rho}$), the algorithm Verify outputs 1. In Fig. 6, Verify outputs 1 if and only if $\forall i \in [0,1], \mathsf{VDPF.Verify}(\tau_i^{(0)}, \tau_i^{(1)}) = 1$ and $\forall i \in [2,3], \tau_i^{(0)} = \tau_i^{(1)}$. The equality of the former follows from the completeness of VDPF. The equality of $\tau_2^{(0)} = \tau_2^{(0)}$ follows from $\eta \in [\alpha \cdot \ell, \alpha \cdot \ell + \ell)$. For the the equality of $\tau_3^{(0)} = \tau_3^{(1)}$, observe that $\mathsf{rs}_{\alpha,\rho} \wedge \bigoplus_{i\in\mathbb{Z}_{N\ell}} [\![\tilde{y}_i]\!] = \mathsf{rs}_{\alpha,\rho} \wedge [\![\beta]\!] = [\![0]\!]$. Hence, it holds that $\tau_3^{(0)} = \tau_3^{(1)}$, as required.

Privacy. We construct an efficient simulator Sim for the view of any verifier P_b. We use the efficient simulator $\mathsf{Sim}_{\mathsf{VDPF}}$ to generate the view of the VDPF output. On input $(1^\lambda, b, \Lambda, X)$, the simulator Sim proceeds as follows:

1 $(f^{(b)}, \tau_0^{(1-b)}) \leftarrow \mathsf{Sim}_{\mathsf{VDPF}}(1^\lambda, b, \mathcal{F}, \mathbb{Z}_N)$

2 $(\tilde{f}^{(b)}, \tau_1^{(1-b)}) \leftarrow \mathsf{Sim}_{\mathsf{VDPF}}(1^\lambda, b, \mathcal{F}', \mathbb{Z}_{N\ell})$

3 Compute $\tau_2^{(b)}, \tau_3^{(b)}$ using Λ, $f^{(b)}$ and $\tilde{f}^{(b)}$ according to VEval described in Fig. 6

4 $\tau_2^{(1-b)} := \tau_2^{(b)}, \tau_3^{(1-b)} := \tau_3^{(b)}$

5 Output $(f^{(b)}, \pi^{(b)} := \tilde{f}^{(b)}, \tau^{(1-b)} := \{\tau_i^{(1-b)}\}_{i\in[0,3]})$

The output distribution of Sim is computationally indistinguishable from the distribution of the real view of P_b, because: (1) the function share $f^{(b)}$, proof share $\pi^{(b)}$ and audit tokens $\tau_0^{(1-b)}$, $\tau_1^{(1-b)}$ are output by the simulator $\mathsf{Sim}_{\mathsf{VDPF}}$, which guarantees the computational indistinguishability; (2) the audit tokens $\tau_2^{(1-b)}$, $\tau_3^{(1-b)}$ are equal to $\tau_2^{(b)}$, $\tau_3^{(b)}$ in the real view due to the CDPF completeness.

Soundness. Suppose for contradiction that there exists a PPT adversary $\mathcal{A}$ and a non-negligible function δ such that

$$\Pr[\mathbf{G}^{\mathsf{sound}}_{\mathsf{CDPF},\mathcal{A}}(\lambda) = 1] \geq \delta(\lambda)$$

Extract $f, \tilde{f}$ from the output by $\mathcal{A}$ in $\mathbf{G}^{\mathsf{sound}}_{\mathsf{CDPF},\mathcal{A}}(\lambda)$. As described in Fig. 6, it holds that $R(\cdot,\cdot) \equiv 1$ in $\star$-CDPF scheme, which implies $T := \bigcup_{i\in\mathbb{Z}_N} \mathcal{Q}_i$. Hence, to win the game $\mathbf{G}^{\mathsf{sound}}_{\mathsf{CDPF},\mathcal{A}}$, the adversary $\mathcal{A}$ need to make Verify output 1 and $\forall i, f \notin \mathcal{Q}_i$. In Fig. 6, Verify outputs 1 if and only if (1) $\forall i \in [0,1], \mathsf{VDPF.Verify}(\tau_i^{(0)}, \tau_i^{(1)}) = 1$, (2) $\tau_2^{(0)} = \tau_2^{(1)}$, and (3) $\tau_3^{(0)} = \tau_3^{(1)}$. For (1), by the soundness property of VDPF, we can assume $f, \tilde{f}$ encodes point functions $f_{\alpha,\beta}, f_{\eta,\beta'}$, respectively. Let $\{y_i\}, \{\tilde{y}_i\}$ be evaluation results of VDPF keys of $f_{\alpha,\beta}, f_{\eta,\beta'}$. For (2), if $\eta \notin [\alpha\cdot\ell, \alpha\cdot\ell+\ell)$ or $\beta \neq \beta'$, which means that $\exists i \in \mathbb{Z}_N$ such that

$$y_i^{(0)} \oplus (\bigoplus_{j\in\mathbb{Z}_\ell} \tilde{y}^{(0)}_{i\cdot\ell+j}) \neq y_i^{(1)} \oplus (\bigoplus_{j\in\mathbb{Z}_\ell} \tilde{y}^{(1)}_{i\cdot\ell+j}),$$

the adversary $\mathcal{A}$ should find a string s such that for an adversarially chosen $b \in \{0,1\}$, for $i \in \mathbb{Z}_N$, $\delta_i := y_i^{(b)} \oplus (\bigoplus_{j\in\mathbb{Z}_\ell} \tilde{y}^{(b)}_{i\cdot\ell+j})$, it holds that

$$H(s) = H(\delta_0 || \ldots || \delta_{N-1})$$

but $s \neq \delta_0 || \ldots || \delta_{N-1}$. It contradicts the collision resistance of H. Hence, we can assume $\eta \in [\alpha\cdot\ell, \alpha\cdot\ell+\ell)$ and $\beta = \beta'$. For (3), let $\rho := \eta - \alpha\ell$. If $\nexists i, f \in \mathcal{Q}_i$, i.e., $\beta \wedge \mathsf{rs}_{\alpha,\rho} \neq 0$, the adversary $\mathcal{A}$ should find a string s' such that for an adversarially chosen $b \in \{0,1\}$, it holds that

$$H(s') = H(\bigoplus_{i\in\mathbb{Z}_N, j\in\mathbb{Z}_\ell} \mathsf{rs}_{i,j} \wedge \widetilde{y}^{(b)}_{i\cdot\ell+j})$$

but $s' \neq \bigoplus_{i\in\mathbb{Z}_N, j\in\mathbb{Z}_\ell} \mathsf{rs}_{i,j} \wedge \widetilde{y}^{(b)}_{i\cdot\ell+j}$. It contradicts the collision resistance of H. □