Fine-grained Policy Constraints for Distributed Point Function

Keyu Ji
jikeyu@zju.edu.cn

Bingsheng Zhang
bingsheng@zju.edu.cn

Kui Ren
kuiren@zju.edu.cn

Abstract

Recently, Servan-Schreiber *et al.* (S&P 2023) proposed a new notion called private access control lists (PACL) for function secret sharing (FSS), where the FSS evaluators can ensure that the FSS dealer is authorized to share the given function with privacy assurance. In particular, for the secret sharing of a point function $f_{\alpha,\beta}$, namely distributed point function (DPF), the authors showed how to efficiently restrict the choice of α via a specific PACL scheme from verifiable DPF. In this work, we show their scheme is insecure due to the lack of assessment of β, and we fix it using an auxiliary output. We then propose more fine-grained policy constraints for DPF. Our schemes allow an attribute-based access control w.r.t. α, and a template restriction for β. Furthermore, we show how to reduce the storage size of the constraint representation from $O(N)$ to $O(\log N)$, where N is the number of constraints. Our benchmarks show that the amortized running time of our attribute-based scheme and logarithmic storage scheme is 2.5× - 3× faster than the state-of-the-art with 2^{15} constraints.

1 Introduction

Function secret sharing (FSS) is first introduced by Boyle *et al.* [1]. The FSS dealer generates and distributes the secret shares of a function f to multiple evaluators. The FSS evaluators can locally evaluate $f(x)$ on any common input x. Distributed point function (DPF) [2] is a useful FSS instance for the function family of point functions, which is usually used to privately search and update distributed data. A variety of important applications have benefited from the efficient realization of DPF, including but not limited to distributed ORAM [3–5], private heavy hitters [6], and privacy-preserving machine learning [7–9].

On top of function privacy, one may also want to restrict the set of functions to be evaluated in practice. That is, only authorized users can obliviously access the specific data. There have been several access control solutions from the literature on anonymous communication [10–12]. But their methods are tailor-made for the specific application. Recently, Servan-Schreiber *et al.* [13] propose a new notion called private access control lists (PACL) to abstract the access control for FSS. They provide two practical PACL schemes for DPF, named the DPF-PACL and the VDPF-PACL from verifiable DPF (VDPF). However, we observe that their VDPF-PACL ensures security only if the function output is restricted to a pre-fixed public value. It results in significant limitations on the scope of applications and in fact even can not support anonymous communication, where the function output is determined by user's message. More specifically, a point function $f_{\alpha,\beta}$ is defined to be the function f such that $f(\alpha) = \beta$ and $f(x) = 0$ for $x \neq \alpha$. We refer to α as the special input

of $f_{\alpha,\beta}$, and refer to β as the special value. In VDPF-PACL, the point functions with the same special input α are mapped to a public verification key vk_α and a secret key sk_α that is the discrete logarithm of vk_α. The verifiers use function shares to evaluate $f_{\alpha,\beta}$ on the list of verification keys and obtain the secret shares of $[\![\mathsf{vk}_\alpha\beta]\!]$. The authors claims that the user without knowledge of sk_α can not pass the verification for any point function with the special input α. Unfortunately, when the special output β is freely selected by the user, we find a malicious adversary who can break their protocol by setting $\beta := \mathsf{vk}_\alpha / g^r$ for any forged secret key r.

Furthermore, in many scenarios, one may do not know the exact identities of all authorized users. Instead, it is able to describe them based on descriptive attributes or credentials. For instance, considering an e-Healthy system, healthcare professionals want to access electronic medical records for diagnosis or clinical research. Since these records contain a large amount of private information, patients may hope to restrict users only with the doctors in specific hospitals and/or the medical researchers in research centers can access their records, without knowing the personal identities of those people. Unfortunately, we note that neither the DPF-PACL nor the VDPF-PACL of [13] can be trivially extended to support the class of attribute-based access control due to their lack of protection against users' collusion attacks. That is, if the secret key sk_α consists of multiple subkeys $\{\mathsf{sk}_{\alpha,i}\}$ according to multiple required attributes, a group of users, each with some attributes, can join together to reconstruct the valid secret key and then pass the PACL verification.

In this work, we fix the VDPF-PACL in [13], and extend the PACL to more fine-grained policy constraints (PC) for DPF, which imply a series of PC constructions for complex FSS classes derived from DPF [8, 14]. Our solutions mainly focus on 2-party DPF, and they can be generalized to multi-party cases as in [13]. In addition, we observe that as the constraint number grows larger, the key selection step becomes the performance bottleneck. To improve efficiency, we propose a new technique to reduce the size of the representation of constraints from linear to logarithmic.

Our contributions. As mentioned above, we first formally show the VDPF-PACL proposed by [13] is insecure against a malicious user. We then propose a fix using VDPF with auxiliary output. Namely, given a point function $f_{\alpha,\beta}$, we transform it to a new point function $f_{\alpha,(\beta,1)}$, and secretly share $f_{\alpha,(\beta,1)}$ via VDPF technique instead of $f_{\alpha,\beta}$. Noice that, the first coordinate of the new function output is equal to the output of $f_{\alpha,\beta}$ for applications, and second coordinate is either 0 or 1 which enables the verifiers to jointly validate the access control.

Next, we propose several schemes for DPF with more fine-grained policy constraints. Our schemes black-box use VDPF, and they support attribute-based access control w.r.t. the special input α and template restriction for the special output β. The latter one is handy in the case of constraint writing, e.g., signature templates. For readability, we present the above two types of policy constraints separately; nevertheless, we emphasize that they can be easily composed. In particular, our attribute-based PC scheme is inspired by the attribute-based encryption (ABE) [15–18]. It can handle constraints in disjunctive normal form (DNF), in which the literals in DNFs represent the attributes of users.

In our template-based PC scheme, each constraint divides the output of the associated functions into free bits and restricted bits. We say a function satisfies the policy constraint iff all restricted bits of its output equal to 0; while the free bits can be chosen arbitrarily. Our construction is based on the fact that AND operation $\wedge$ is distributive over XOR operation $\oplus$; therefore, the verifiers can locally computes the XOR secret shares of the AND result between the XOR-shared function output and the restraint string.

Moreover, we introduce a new DPF primitive named incremental VDPF (IVDPF). It allows two evaluators obliviously obtain the (scaled) bit-decomposition of the special input of the shared point function. Based on IVDPF, we construct a PC scheme that improves the DPF-PACL in [13]. It reduces both the storage size of the constraint representation and the computational complexity of the key selection from $O(N)$ to $O(\log N)$, where N is the number of constraints.

In Section 7, we show the performance of our scheme is much more efficient than the state-of-the-art PACLs [13]; our attribute-based scheme reduce the amortized running time of VDPF-PACL by 2× - 4×. In addition, when the number of constraints is large, our IVDPF-PC scheme significantly improves the performance compared to DPF-PACL. For instance, the verification speed of IVDPF-PC is 2.5× faster than DPF-PACL with 2^{15} constraints.

Related work. Recent anonymous communication systems usually utilize DPF to enable users privately write messages [10–12, 19]. To prevent private data against malicious users, some of them design ad-hoc methods of access control for DPF. The "mailbox" system Express [10] assigns λ-bit virtual addresses 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 [11] 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 [12] constructs an anonymous broadcasting system. It proposes a new access control mechanism via secret-shared Carter-Wegman MAC [20]. That is, servers holds a secret-shared MAC tag from a unique secret-shared message for each broadcast channel. Servers only allow the client who knows the valid MAC tag to write messages to the corresponding channel via DPF.

Servan-Schreiber *et al.* [13] abstracts PACL for FSS from these 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.

2 Preliminary

Notations. Let λ be the security parameter. Let $[\![x]\!]$ denote the additively secret sharing of an object x. $[\![x]\!] \leftarrow \mathsf{AddShare}_{\mathbb{G},n}(x)$ stands for generating n additive shares of x, where $[\![x]\!] := \{x^{(0)}, \ldots, x^{(n-1)}\}$ and $x = \sum_{i\in\mathbb{Z}_n} x^{(i)}$ over $\mathbb{G}$. Let $\langle x\rangle$ denote the multiplicatively secret sharing of an object x. $\langle x\rangle \leftarrow \mathsf{MulShare}_{\mathbb{G},n}(x)$ 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 $\mathbb{G}$.

Function Secret Sharing. An FSS scheme includes two algorithms $(\mathsf{Gen}, \mathsf{Eval})$. It allows a dealer to generate 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 about f. DPF [2] is a useful FSS instance for the function family of point functions. In order to achieve malicious security for DPF, researchers provide a primitive, *verifiable* DPF (VDPF) [6, 21]. It is a special DPF that additionally allows evaluators to check whether the dealer's inputs are well-formed point function. The VDPF scheme is defined as follows:

Definition 1 (Verifiable Distributed Point Function [21]). *A 2-party VDPF scheme, parameterized by a function family $\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{G}\}$ of point functions, 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.*
- $1/0 \leftarrow \mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$ *is the verification algorithm that takes input as a pair of tokens. It outputs 1 for accept or 0 for reject.*

A secure VDPF must satisfy three properties as follows:

- ***Correctness.*** *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 $\mathsf{View}_{\mathsf{VDPF}}(b, f, X)$ 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:*

$$\mathsf{View}_{\mathsf{VDPF}}(b, f, X) \approx_c \mathsf{Sim}(1^\lambda, b, \mathcal{F}, X)$$

- ***Soundness.*** *There exists a negligible function negl such that for any $X \subseteq \{0,1\}^n$, and $(\{y^{(b)}_x\}_{x\in X}, \tau^{(b)}) \leftarrow \mathsf{Eval}(b, f^{(b)}, X)$ for $b \in \{0,1\}$, it holds that*

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

Disjunctive normal form. DNF is a disjunction of conjunctions of literals. A k-DNF is a DNF in which every conjunction is of size k. In our paper, each literal x indicates a user's attribute a_x, and $\neg x$ corresponds to a different and contradictory attribute $a_{x'}$ from a_x.

Bilinear Maps and the 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 [15,17] 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. Following the definition in [21], we say a function family $\mathcal{H}$ is XOR-collision resistant if no PPT adversary given a randomly sampled $h \leftarrow \mathcal{H}$ can find four values x_0, x_1, x_2, x_3 such that $\{x_0, x_1\} \neq \{x_2, x_3\}$ and $h(x_0) \oplus h(x_1) = h(x_2) \oplus h(x_3) \neq 0$ with more than a negligible probability.

3 Policy Constraints for FSS

In this section, we present a formal definition of policy constraints for FSS, which can be viewed as a generalization of PACL proposed by [13]. Let $\mathcal{F}$ be a function family. The policy constraints for $\mathcal{F}$ consists of a constraint list $\Lambda := \{\Lambda_i\}_{i \in \mathbb{Z}_N}$ and an efficiently computable predicate Check. Each constraint Λ_i determines a set of authorized functions $\mathcal{Q}_i \subseteq \mathcal{F}$, and $\mathcal{Q}_i \cap \mathcal{Q}_j = \emptyset$ for any $i \neq j$. More specifically, the constraint Λ_i contains ℓ verification keys $\{\mathsf{vk}_{i,j}\}_{j \in \mathbb{Z}_\ell}$ associated with the subsets $\{\mathcal{Q}_{i,j}\}_{j \in \mathbb{Z}_\ell}$ respectively, such that $\bigcup_{j \in \mathbb{Z}_\ell} \mathcal{Q}_{i,j} = \mathcal{Q}_i$. Given a function f and a secret key sk, we have $\mathsf{Check}(\Lambda, f, \mathsf{sk}) = 1$ if and only if f belongs to an authorized subset, say $\mathcal{Q}_{i,j}$, and $(\mathsf{vk}_{i,j}, \mathsf{sk})$ belongs to a relation R. Note that, N, ℓ are integers in $\mathsf{poly}(\lambda)$. In this paper, we focus on PC for 2-party FSS where two verifiers securely attest $\mathsf{Check} = 1$ over the secret-shared function and the secret key, and define it as follows:

Definition 2 (Policy Constraints for 2-party FSS)**.** *Let $\mathcal{F}$ be a function family. Suppose* $(\mathsf{Gen}, \mathsf{Eval})$ *instantiate a 2-party FSS scheme for $\mathcal{F}$. A PC scheme for 2-party FSS consists of four PPT algorithms* $(\mathsf{PolicyGen}, \mathsf{Prove}, \mathsf{Audit}, \mathsf{Verify})$*:*

- $(\Lambda, \xi) \leftarrow \mathsf{PolicyGen}(1^\lambda, \mathcal{F})$ *is the policy generation algorithm that takes input as the security parameter 1^λ and the description of a function family $\mathcal{F}$. It determines the authorized function sets $\{\mathcal{Q}_i \subseteq \mathcal{F}\}_{i \in \mathbb{Z}_N}$, and outputs a constraint lists $\Lambda := \{\Lambda_i\}_{i \in \mathbb{Z}_N}$ and a trapdoor ξ. Anyone holding ξ is able to generate valid secret keys for Λ.*
- $\mathsf{sk} \leftarrow \mathsf{SKGen}(1^\lambda, \mathsf{id}, \xi)$ *is the secret key generation algorithm that takes input as the security parameter 1^λ, a policy index* id[1] *and a trapdoor ξ. It outputs a secret key* sk*.*
- $(\pi^{(0)}, \pi^{(1)}) \leftarrow \mathsf{Prove}(\Lambda, f, \mathsf{sk})$ *is the proof generation algorithm that takes input as a constraint list Λ, function $f \in \mathcal{F}$ and a secret key* sk*. It outputs the proof shares $(\pi^{(0)}, \pi^{(1)})$. If it is clear in the context, we denote $[\![\pi]\!] := (\pi^{(0)}, \pi^{(1)})$.*
- $\tau^{(b)} \leftarrow \mathsf{Audit}(b, \Lambda, f^{(b)}, \pi^{(b)})$ *is the audit algorithm that takes input as an index $b \in \{0, 1\}$, a constraint list Λ, a function share $f^{(b)}$ and a proof share $\pi^{(b)}$. It outputs an audit token $\tau^{(b)}$.*
- $1/0 \leftarrow \mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$ *is the verification algorithm that takes input as two tokens. It outputs 1 for accept or 0 for reject.*

[1] The representation of id depends on the concrete system, e.g. integers for PACL [13] and attributes for our attributed-based PC scheme.

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 PC scheme is secure if it satisfies three properties as follows:

- ***Correctness.*** *For any* $(\Lambda, \xi) \leftarrow \mathsf{PolicyGen}(1^\lambda, \mathcal{F})$*, function* $f \in \mathcal{F}$ *and secret key* $\mathsf{sk} \leftarrow \mathsf{SKGen}(1^\lambda, \mathsf{id}(f, \Lambda), \xi)$[2]*, we have*

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

- ***Privacy.*** *For a constraint list* Λ*, a function* $f \in \mathcal{F}$ *and a secret key* sk*, define the view* $\mathsf{View}_{\mathsf{PC}}(b, \Lambda, f, \mathsf{sk})$ *as the probability distribution ensemble* $\{(\pi^{(b)}, \tau^{(1-b)})\}_\lambda$*, where* $(\pi^{(0)}, \pi^{(1)}) \leftarrow \mathsf{Prove}(\Lambda, f, \mathsf{sk})$ *and* $\tau^{(1-b)} \leftarrow \mathsf{Audit}(\Lambda, f^{(1-b)}, \pi^{(1-b)})$*. There exists a PPT simulator* Sim *such that for all* $f \in \mathcal{F}$ *and* sk*, the following two distributions are computationally indistinguishable:*

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

- ***Soundness.*** *We say PC is sound, if for any PPT adversary* $\mathcal{A}$ *with oracle access to* GetKey*, it holds that*

$$\Pr[\mathsf{Soundness}_{\mathcal{A},b}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where $\mathsf{Soundness}_{\mathcal{A},b}(\lambda)$ *is depicted in Fig. 1.*

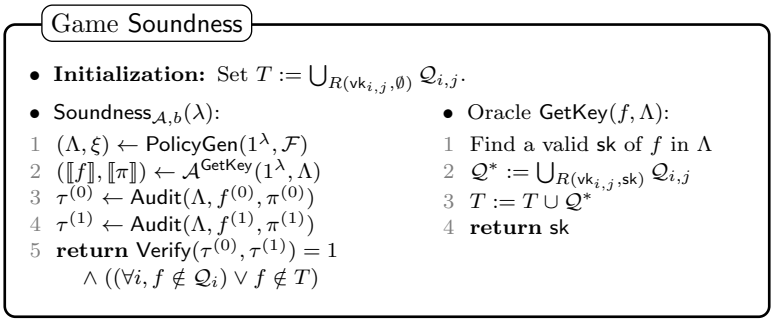

Figure 1: PC soundness game.

Remark 1. *We note that the privacy definition of [13] is problematic and does not capture the practice. 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 [13], the simulator is given the adversary's message as input, which deviates from the reality.*

[2]Here we abuse the notation id as a function that outputs the policy index corresponding to f.

3.1 Threat model

Our PC system involves the following entities:

- the data owner who determines the policy constraints by $\mathsf{PolicyGen}$ and outsources data to servers,
- the user (i.e., the prover) who wants to access data and proves it satisfies the constraint by Prove, and
- the servers (i.e., the verifiers) securely verify the access proof using Audit and Verify.

In a real-world application, the user needs a valid secret key as authorization to pass the verification. We assume there exists a trusted authority to issue the secret keys.

In this paper, we consider the malicious security for a single corrupted party among one prover and two verifiers. More specifically, as described in Definition 2, a secure PC scheme for 2-party FSS protects function privacy as long as one of the two servers is honest, and guarantees soundness against any malicious users.

4 Revisiting VDPF-PACL [13]

The recent work [13] presents PACL schemes for the class of (verifiable) DPFs. In this section, we show their VDPF-PACL scheme is not sound.

The VDPF-PACL scheme. The VDPF-PACL scheme assumes $\mathbb{G} := \mathbb{F}_p^*$ is a group with order $p-1$ and generator g in which the discrete logarithm problem is computationally intractable. Without loss of generality, let $\ell = 1$ and focus on 2-party VDPF. Given a family $\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{F}_p\}$ of point functions, the VDPF-PACL initiates constants:

- $\mathsf{PolicyGen}(1^\lambda, \mathcal{F})$:

1 $N := 2^n$
2 **for** $i \in \mathbb{Z}_N$:
3 $\quad \mathsf{sk}_i \leftarrow \mathbb{Z}_{p-1}$, $\mathsf{vk}_i := g^{\mathsf{sk}_i}$
4 $\Lambda_i := \mathsf{vk}_i, \xi_i := \mathsf{sk}_i$
5 **return** $\Lambda := \{\Lambda_i\}_{i\in\mathbb{Z}_N}, \xi := \{\xi_i\}_{i\in\mathbb{Z}_N}$

In [13], 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 as input the discrete logarithm x of w base g, 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 as input the additive share of w and the SPoSS proof share, outputs a verification token; (3) $\mathsf{SPoSS.Verify}(\tau^{(0)}, \tau^{(1)})$ takes as input the tokens of two verifier, outputs 1 if and only if $w^{(0)} + w^{(1)} = g^x$ over the field $\mathbb{F}_p$.

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:

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

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 select 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 $(\{y^{(b)}_i\}_{i\in\mathbb{Z}_N}, \tau^{(b)}_0) \leftarrow \mathsf{VDPF.Eval}(b, f^{(b)}_{\alpha,\beta}, \mathbb{Z}_N)$

2 $w^{(b)} := \sum_{i=0}^{N-1} \Lambda_i \cdot y^{(b)}_i$

3 $\tau^{(b)}_1 \leftarrow \mathsf{SPoSS.Audit}(b, w^{(b)}, \pi^{(b)})$

4 **return** $\tau^{(b)} := (\tau^{(b)}_0, \tau^{(b)}_1)$

Finally, two verifiers 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 $\tau^{(0)} := (\tau^{(0)}_0, \tau^{(0)}_1)$

2 Parse $\tau^{(1)} := (\tau^{(1)}_0, \tau^{(1)}_1)$

3 **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.

Attack description. 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 $\mathbb{F}_p$. 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:

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

1 $\alpha \leftarrow \{0,1\}^n$, $r \leftarrow \mathbb{Z}_{p-1}$

2 $\beta := g^r \cdot \mathsf{vk}_\alpha^{-1} \pmod p$

3 $[\![f_{\alpha,\beta}]\!] \leftarrow \mathsf{VDPF.Gen}(1^\lambda, f_{\alpha,\beta})$

4 $[\![\pi]\!] \leftarrow \mathsf{Prove}(\Lambda, f_{\alpha,\beta}, r)$

5 **return** $([\![f_{\alpha,\beta}]\!], [\![\pi]\!])$

Using the VDPF keys $[\![f_{\alpha,\beta}]\!]$, two verifiers obliviously obtains 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 $[\![f_{\alpha,\beta}]\!]$ 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 $\beta \in \mathbb{F}_p$.

Our fix. Our attack is based on this 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 verifiers additionally check $\beta = 1$, our attack is fixed. However, in many applications, one may need to use the customized β. For example, in anonymous communication systems [10–12, 19], the value of β is determined by the message that the user wants to send. 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 the second item 1 of this tuple is named the special auxiliary output. We let user secret-share the new point function $f_{\alpha,(\beta,1)}$ via VDPF to be evaluated. After that, the verifiers use the auxiliary outputs to select the target verification key, and then jointly check if the special auxiliary output is equal to 1. More specifically, we adapted the algorithms 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 Parse $y_i^{(b)} := (y_{i,0}^{(b)}, y_{i,1}^{(b)}) \in \mathbb{Z}_p^2$ for $i \in \mathbb{Z}_N$

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

where $H := \{0,1\}^* \to \{0,1\}^\lambda$ be 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 Fine-grained PC schemes for DPF

In this section, we propose the fine-grained PC schemes for 2-party DPF. We adopt the verifiable DPF with auxiliary output as described above. Similar to [13], our PC schemes are applicable to complex FSS classes derived from DPFs.

5.1 Attribute-based policy constraints

One common approach of all previous DPF access control schemes is that they view the secret key as a fixed private string. Therefore, when the system aims to grant the access right of the function f to a user with multiple certain credentials or attributes, the previous

schemes fail to distinguish whether these attributes are (jointly) held by a group of colluding users or the valid individuals.

We propose an attribute-based PC (Attr-PC) construction to resolving this drawback. Our concept draws inspiration from the ABE technique of [15–18]. At a high level, we make each constraint be an m-DNF with ℓ conjunctions, and each conjunction is considered as a verification key. The literals in DNFs indicate the attributes of users. The verifiers first obliviously obtain a secret-shared conjunction in the target constraint, which is equivalent to an ABE ciphertext. The verifiers then use the proof shares to jointly "decrypt" the selected ciphertext and check if the decryption is successful without revealing the prover's attributes.

The challenge of this approach is how to securely "decrypt" a secret-shared ciphertext. The standard ABE scheme is usually constructed by a bilinear map e. Naively, the verifiers are required to evaluate the bilinear map over two secret-shared arguments in MPC, such as $e(\langle E\rangle, \langle A\rangle)$. However, this would result multiple exchange of messages between the verifiers. For this issue, we let the prover locally compute and distribute the secret shares of the bilinear map result because it knows both arguments E, A. To prevent the prover from cheating through such assistance, we introduce a multi-verifier ZK proof of knowledge (ZK-PoK) to ensure the prover holds the corresponding witness matching the computed bilinear map result.

5.1.1 Multi-verifier ZK-PoK

It is a non-interactive proof between a prover and two verifiers. Let e be a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$, where $\mathbb{G}, \mathbb{G}_t$ have the prime order p and g is the generator of $\mathbb{G}$. The verifier P_b holds secret shares $E^{(b)} \in \mathbb{G}, k^{(b)} \in \mathbb{G}_t$ for $b \in \{0, 1\}$. The prover holds group elements $E := E^{(0)} \cdot E^{(1)}, A \in \mathbb{G}$. The prover wants to convince the verifiers that $k^{(0)} \cdot k^{(1)} = e(E^{(0)} \cdot E^{(1)}, A)$.

Definition 3 (Multi-verifier ZK-PoK)**.** *A non-interactive 2-verifier ZK proof of knowledge over the secret shares $\langle E\rangle \in \mathbb{G}, \langle k\rangle \in \mathbb{G}_t$ consists of three PPT algorithms* (Prove, Audit, Verify) *defined as follows:*

- $(\pi^{(0)}, \pi^{(1)}) \leftarrow \mathsf{Prove}(A, E)$ *is the proof generation algorithm that takes input as two group elements $A, E \in \mathbb{G}$. It outputs proof shares $(\pi^{(0)}, \pi^{(1)})$. If it is clear in the context, we denote $\langle\pi\rangle := (\pi^{(0)}, \pi^{(1)})$.*
- $\tau^{(b)} \leftarrow \mathsf{Audit}(b, \pi^{(b)}, E^{(b)}, k^{(b)})$ *is the audit algorithm that takes input as an index $b \in \{0, 1\}$, a proof share $\pi^{(b)}$ and multiplicative shares $E^{(b)} \in \mathbb{G}, k^{(b)} \in \mathbb{G}_t$. It outputs an audit token.*
- $1/0 \leftarrow \mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$ *is the verification algorithm that takes input as two audit tokens. It outputs 1 for accept or outputs 0 for reject.*

We say a protocol is a secure ZK-PoK if it satisfies the following properties:

Construction ZK-PoK

- **Parameters**: Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$ be a bilinear map. The prime number p is the order of $\mathbb{G}, \mathbb{G}_t$ and g is the generator of $\mathbb{G}$. Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function that we model as a random oracle.

- $\mathsf{Prove}(A, E \in \mathbb{G})$:

$(A^{(0)}, A^{(1)}) \leftarrow \mathsf{MulShare}_{\mathbb{G},2}(A)$
$(x^{(0)}, x^{(1)}, y^{(0)}, y^{(1)}, z^{(0)}, z^{(1)}) \leftarrow \mathsf{Beaver}_{\mathbb{Z}_p,2}$
$x := x^{(0)} + x^{(1)}$
$y := y^{(0)} + y^{(1)}$
$s^{(0)}, s^{(1)} \leftarrow \{0,1\}^\lambda$ //Random nonces
$r^{(0)} := H(s^{(0)}, A^{(0)}, x^{(0)}, y^{(0)}, z^{(0)})$
$r^{(1)} := H(s^{(1)}, A^{(1)}, x^{(1)}, y^{(1)}, z^{(1)})$
$r := r^{(0)} \oplus r^{(1)}$
$B := A \cdot g^{-x} \in \mathbb{G}$
$C := E^r \cdot g^{-y} \in \mathbb{G}$
$\pi^{(0)} := (A^{(0)}, B, C, x^{(0)}, y^{(0)}, z^{(0)}, r, s^{(0)})$
$\pi^{(1)} := (A^{(1)}, B, C, x^{(1)}, y^{(1)}, z^{(1)}, r, s^{(1)})$
return $(\pi^{(0)}, \pi^{(1)})$

- $\mathsf{Audit}(b, \pi^{(b)}, E^{(b)} \in \mathbb{G}, k^{(b)} \in \mathbb{G}_t)$:

Parse $\pi^{(b)} := (A^{(b)}, B, C, x^{(b)}, y^{(b)}, z^{(b)}, r, s^{(b)})$
$\hat{r}^{(b)} := H(s^{(b)}, A^{(b)}, x^{(b)}, y^{(b)}, z^{(b)})$
$D^{(b)} := e(C,B)^b \cdot e(C,g)^{x^{(b)}} \cdot e(g,B)^{y^{(b)}} \cdot e(g,g)^{z^{(b)}} \cdot (k^{(b)})^{-r}$
$T_0^{(b)} := A^{(b)} \cdot g^{-x^{(b)}}$, $T_1^{(b)} := (E^{(b)})^r \cdot g^{-y^{(b)}}$
return $\tau^{(b)} := (T_0^{(b)}, T_1^{(b)}, D^{(b)}, \hat{r}^{(b)}, r, B, C)$

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

Parse $\tau^{(0)} := (T_0^{(0)}, T_1^{(0)}, D^{(0)}, \hat{r}^{(0)}, r, B, C)$
Parse $\tau^{(1)} := (T_0^{(1)}, T_1^{(1)}, D^{(1)}, \hat{r}^{(1)}, r, B, C)$
return $\hat{r}^{(0)} \oplus \hat{r}^{(1)} = r \wedge T_0^{(0)} \cdot T_0^{(1)} = B \wedge T_1^{(0)} \cdot T_1^{(1)} = C \wedge D^{(0)} \cdot D^{(1)} = 1$

Figure 2: Multi-verifier ZK proof of knowledge.

- ***Completeness.*** *For all $E, A \in \mathbb{G}$ and $k := e(E, A)$, it holds that*

$$\Pr\left[\begin{array}{l}(E^{(0)}, E^{(1)}) \leftarrow \mathsf{MulShare}_{\mathbb{G}}(E);\\ (k^{(0)}, k^{(1)}) \leftarrow \mathsf{MulShare}_{\mathbb{G}_t}(k);\\ (\pi^{(0)}, \pi^{(1)}) \leftarrow \mathsf{Prove}(A, E);\\ \tau^{(0)} \leftarrow \mathsf{Audit}(0, \pi^{(0)}, E^{(0)}, k^{(0)});\\ \tau^{(1)} \leftarrow \mathsf{Audit}(1, \pi^{(1)}, E^{(1)}, k^{(1)}) :\\ \mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1\end{array}\right] = 1$$

- ***Knowledge soundness.*** *If there exists a PPT adversary $\mathcal{A}$ such that for all group elements $E \in \mathbb{G}, k \in \mathbb{G}_t$, $\mathcal{A}$ produces $\langle \pi^* \rangle$ such that $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1$ with probability $\delta(\lambda)$, then there exists an efficient knowledge extractor $\mathcal{E}$ and negligible function negl, such that*

$$\Pr\left[\begin{array}{l} A \leftarrow \mathcal{E}^{\mathcal{A}}(E, k) :\\ k = e(E, A)\end{array}\right] \geq \delta(\lambda) - \mathsf{negl}(\lambda)$$

- ***Zero knowledge.*** *For group elements $A, E \in \mathbb{G}$ and $k := e(E, A)$, define the view $\mathsf{View}_{\mathsf{ZK\text{-}PoK}}(b, A, E)$ as the probability distribution ensemble $\{(\pi^{(b)}, \tau^{(1-b)})\}_\lambda$, where $(\pi^{(0)}, \pi^{(1)}) \leftarrow \mathsf{Prove}(A, E)$ and $\tau^{(1-b)} \leftarrow \mathsf{Audit}(1-b, \pi^{(1-b)}, E^{(1-b)}, k^{(1-b)})$. There exists a PPT simulator Sim such that for all $E, A \in \mathbb{G}$ and $k := e(E, A)$, the following two distributions are statistically indistinguishable:*

$$\mathsf{View}_{\mathsf{ZK\text{-}PoK}}(b, A, E) \approx \mathsf{Sim}(1^\lambda, b, \mathbb{G}, \mathbb{G}_t)$$

We describe our construction in Fig. 2. It leverages the bilinearity of the bilinear map to check the consistency of the arguments and result. That is, suppose $A := g^\alpha, E := g^\beta$ and $k := e(g,g)^\gamma$, the verifiers are able to check whether $e(\langle A \rangle, \langle E \rangle) = \langle k \rangle$ by verifying $[\![\alpha \cdot \beta - \gamma]\!] = 0$ in the exponent basing $e(g,g)$. Naively, the prover first generates the

multiplicative secret shares of $\langle A \rangle$, where the exponents of all shares basing g form the additively secret-shared $[\![\alpha]\!]$. To eliminate the interaction of verifiers in the calculation of $\langle e(\langle A \rangle, \langle E \rangle) \rangle := e(g,g)^{[\![\alpha \cdot \beta]\!]}$, we make the prover derives a Beaver triple $([\![x]\!], [\![y]\!], [\![z]\!])$ such that $xy = z$ over the field $\mathbb{Z}_p$ [22], and compute the intermediate results $B := g^{\alpha - x}, C := g^{\beta - y}$. Using the Beaver triple and the intermediate results, each verifier can locally compute the multiplicative secret share of $\langle e(\langle A \rangle, \langle E \rangle) \rangle$ following Beaver's multiplication protocol. More specifically, the verifiers jointly compute $[\![\alpha \cdot \beta]\!] = (\beta - y)(\alpha - x) + (\beta - y)[\![x]\!] + (\alpha - x)[\![y]\!] + [\![z]\!]$ in the exponent basing $e(g,g)$. Later, the verifiers compute $\langle D \rangle := \langle e(g,g)^{\alpha \cdot \beta} \rangle / \langle k \rangle := e(g,g)^{[\![\alpha \cdot \beta - \gamma]\!]}$, and audit the consistency of the intermediate results. Finally, the verifiers exchange their shares to check $D = e(g,g)^0 = 1$ and the validation of intermediate results.

Unfortunately, if a malicious prover shares an invalid multiplication triple (i.e., $xy = z + \Delta$, for some constant $\Delta \neq 0$) to the verifiers, the naive approach above would fail. As described in [23], this attack can be defended against by introducing a random number r (distributed independently to Δ) and verifying $r[\![\alpha\beta]\!] - r[\![\gamma]\!] = 0$. Inspired by SPoSS [13], we exploit Fiat-Shamir transform [24] and let the prover obtain r from a random oracle H with random nonces $s^{(0)}, s^{(1)}$, which is used to mask the other inputs to H [25].

Security. We show the security of the construction described in Fig. 2 with the following theorem, and its proof can be found in Appendix B.1.

Theorem 1. *Let p be a prime chosen with respect to the security parameter λ and let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$ be a bilinear map, where $\mathbb{G}, \mathbb{G}_t$ are two multiplicative cyclic groups with order p. Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a random oracle. The ZK-PoK described in Fig. 2 is secure under the random oracle model.*

5.1.2 Construction using ZK-PoK

Our Attr-PC construction is depicted in Fig. 3. Each constraint is a m-DNF so that each verification key associated with m attributes. We focus on the point function family $\mathcal{F} := \{f : \{0,1\}^n \to \tilde{\mathbb{G}}\}$ and set each $Q_{i,j} := \{f_{i,v} \in \mathcal{F} | v \in \tilde{\mathbb{G}}\}$ for simplicity.

The method $\mathsf{PolicyGen}$ generates a public constraint list Λ for $\mathcal{F}$. More specifically, $\mathsf{PolicyGen}$ selects random $w, t \in \mathbb{Z}_p$ as the trapdoor. For each verification key $\mathsf{vk}_{i,j} \in \Lambda_i$, it determines m required attributes $\{a_d\}_{d \in \mathbb{Z}_m}$, and randomly samples a mask $s_{i,j} \in \mathbb{Z}_p$ and a polynomial q with degree $m-1$, such that $q(0) = s_{i,j}$. Using the polynomial q, it calculates $\{B_{i,j,d} := g^{t \cdot q(d+1) - H'(a_d) \cdot s_{i,j}}\}_{d \in \mathbb{Z}_m}$ associated with m required attributes. In conclusion, $\mathsf{vk}_{i,j} := (C_{i,j} := e(g,g)^{w \cdot s_{i,j}}, E_{i,j} := g^{s_{i,j}}, \{B_{i,j,d}\}_{d \in \mathbb{Z}_m})$, where the first element $C_{i,j}$ is what the prover should prove it can recover (i.e., successfully "decrypt"), and other elements ensure that a prover can recover $C_{i,j}$ if and only if it holds all the required attributes.

We assume that there is a trusted authority (TA) holding the trapdoor $\xi := (w,t)$ of the constraint list Λ, and the TA issues the secret key sk according to user's attributes. The algorithm SKGen in Fig. 3 defines the secret key generation method. The TA computes $L := g^r, K := g^{w+tr}$ using a random mask r, and calculates $A_i := g^{H'(a_i)r}$ for each attribute a_i. Since $\mathsf{sk} := (L, K, \{A_i\})$ is randomized by r, it provides the security against users' collusion attack. Note that, H' is a collision-resistant hash function, thus the probability of $\{H'(a_i)\}_{i \in \mathbb{Z}_m} = \{H'(\tilde{a}_i)\}_{i \in \mathbb{Z}_m}$ is negligible for any attribute sets $\{a_i\}_{i \in \mathbb{Z}_m} \neq \{\tilde{a}_i\}_{i \in \mathbb{Z}_m}$.

As show in algorithm Prove, for a point function $f_{\alpha,\beta}$, the prover first selects the target verification key $\mathsf{vk}_{\alpha,\rho} := (C_{\alpha,\rho}, E_{\alpha,\rho}, \{B_{\alpha,\rho,d}\}_{d \in \mathbb{Z}_m})$ from the public Λ by $f_{\alpha,\beta}$ and sk. It then computes the proof shares using sk and $E_{\alpha,\rho}$. The proof $[\![\pi]\!]$ includes a pair of VDPF keys $[\![f_{\eta,(\beta,1)}]\!]$ where $\eta := \alpha\ell + \rho$ for selecting the verification key $\mathsf{vk}_{\alpha,\rho}$, two elements L, K

Construction Attr-PC

- **Parameters**: Let $\mathcal{F} := \{f : \{0,1\}^n \to \tilde{\mathbb{G}}\}, \mathcal{F}' := \{f : \{0,1\}^{n+\log \ell} \to \tilde{\mathbb{G}} \times \mathbb{Z}_p\}$ be two function families of point functions, and $N := 2^n$. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$ be a bilinear map. The prime number p is the order of $\mathbb{G}, \mathbb{G}_t$ and g is the generator of $\mathbb{G}$. Let $H : \{0,1\}^* \to \{0,1\}^{4\lambda}$ be a hash function sampled from a family $\mathcal{H}$ that is both collision-resistant and XOR-collision-resistant. Let $H' : \{0,1\}^* \to \mathbb{Z}_p$ be a hash function sampled from a family $\mathcal{H}'$ that is collision-resistant. Set Lagrange coefficients as $\Delta_{i,m}(x) := \prod_{j \in [m]/\{i\}} \frac{x-j}{i-j}$.

- $\mathsf{PolicyGen}(1^\lambda, \mathcal{F})$:
  1. $w, t \leftarrow \mathbb{Z}_p$, $\xi := (w, t)$
  2. $W := e(g,g)^w, T := g^t$
  3. **for** $i \in \mathbb{Z}_N, j \in \mathbb{Z}_\ell$:
  4. Choose $\{a_d\}_{d \in \mathbb{Z}_m}$ over the universe of attributes
  5. $s_{i,j} \leftarrow \mathbb{Z}_p, C_{i,j} := W^{s_{i,j}}, E_{i,j} := g^{s_{i,j}}$
  6. Sample a random polynomial q with degree $m-1$, such that $q(0) = s_{i,j}$
  7. **for** $d \in \mathbb{Z}_m$:
  8. $B_{i,j,d} := T^{q(d+1)} \cdot g^{-H'(a_d) \cdot s_{i,j}} \in \mathbb{G}$
  9. $\mathsf{vk}_{i,j} := (C_{i,j}, E_{i,j}, \{B_{i,j,d}\}_{d \in \mathbb{Z}_m})$
  10. $\Lambda_i := \{\mathsf{vk}_{i,j}\}_{j \in \mathbb{Z}_\ell}, \forall i \in \mathbb{Z}_N$
  11. **return** $(\Lambda := \{\Lambda_i\}_{i \in \mathbb{Z}_N}, \xi)$

- $\mathsf{SKGen}(1^\lambda, \{a_i\}_{i \in \mathbb{Z}_m}, \xi)$:
  1. Parse $\xi := (w, t)$
  2. $r \leftarrow \mathbb{Z}_p$, $L := g^r, K := g^{w + t \cdot r}$
  3. **for** $i \in \mathbb{Z}_m$: $A_i := g^{H'(a_i) \cdot r} \in \mathbb{G}$
  4. $\mathsf{sk} := (L, K, \{A_i\}_{i \in \mathbb{Z}_m})$
  5. **return** sk

- $\mathsf{Prove}(\Lambda, f_{\alpha,\beta} \in \mathcal{F}, \mathsf{sk})$:
  1. Select the target $\mathsf{vk}_{\alpha,\rho}$ from Λ which can be successfully "decrypted" by sk.
  2. Parse $\mathsf{vk}_{\alpha,\rho} := (C, E, \{B_d\}_{d \in \mathbb{Z}_m})$
  3. Parse $\mathsf{sk} := (L, K, \{A_d\}_{d \in \mathbb{Z}_m})$
  4. **for** $d \in \mathbb{Z}_m$:
  5. $(A_d^{(0)}, A_d^{(1)}) \leftarrow \mathsf{MulShare}_{\mathbb{G},2}(A)$
  6. $k_d^{(0)} := e(E, A_d^{(0)}), \;\; k_i^{(1)} := e(E, A_d^{(1)})$
  7. $(\varphi_d^{(0)}, \varphi_d^{(1)}) \leftarrow \mathsf{ZK\text{-}PoK.Prove}(A_d, E)$
  8. $\eta := \alpha \cdot \ell + \rho$
  9. $[\![f_{\eta,(\beta,1)}]\!] \leftarrow \mathsf{VDPF.Gen}(1^\lambda, f_{\eta,(\beta,1)} \in \mathcal{F}')$
  10. $\pi^{(0)} := (f^{(0)}_{\eta,(\beta,1)}, L, K, \{k_d^{(0)}, \varphi_d^{(0)}\}_{d \in \mathbb{Z}_m})$
  11. $\pi^{(1)} := (f^{(1)}_{\eta,(\beta,1)}, L, K, \{k_i^{(1)}, \varphi_i^{(0)}\}_{i \in \mathbb{Z}_m})$
  12. **return** $(\pi^{(0)}, \pi^{(1)})$

- $\mathsf{Audit}(b, \Lambda, f^{(b)}_{\alpha,\beta}, \pi^{(b)})$:
  1. Parse each $\mathsf{vk}_{i,j} := (C_{i,j}, E_{i,j}, \{B_{i,j,d}\}_{d \in \mathbb{Z}_m}) \in \Lambda$
  2. Parse $\pi^{(b)} := (f^{(b)}_{\eta,(\beta,1)}, L, K, \{k_d^{(b)}, \varphi_d^{(0)}\}_{d \in \mathbb{Z}_m})$
  3. $(\{y_i^{(b)}\}_{i \in \mathbb{Z}_N}, \tau_0^{(b)}) \leftarrow \mathsf{VDPF.Eval}(b, f^{(b)}_{\alpha,\beta}, \mathbb{Z}_N)$
  4. $(\{\tilde{y}_i^{(b)}\}_{i \in \mathbb{Z}_{N\ell}}, \tau_1^{(b)}) \leftarrow \mathsf{VDPF.Eval}(b, f^{(b)}_{\eta,(\beta,1)}, \mathbb{Z}_{N\ell})$
  5. Parse each $\tilde{y}_i^{(b)} := (\tilde{y}_{i,0}^{(b)}, \tilde{y}_{i,1}^{(b)})$
  6. $\tau_2^{(b)} := H(b + (-1)^b \cdot \sum \tilde{y}_{i,1}^{(b)})$
  7. $\tau_3^{(b)} := \bigoplus_{i \in \mathbb{Z}_N} H((y_i^{(b)} - \sum_{j \in \mathbb{Z}_\ell} \tilde{y}^{(b)}_{i \cdot \ell + j, 0}) || i)$
  8. $C^{(b)} := \prod_{i=0}^{n-1} \prod_{j=0}^{\ell-1} C_{i,j}^{-\tilde{y}^{(b)}_{i \cdot \ell + j, 1}}$
  9. $E^{(b)} := \prod_{i=0}^{n-1} \prod_{j=0}^{\ell-1} E_{i,j}^{-\tilde{y}^{(b)}_{i \cdot \ell + j, 1}}$
  10. **for** $d \in \mathbb{Z}_m$:
  11. $B_d^{(b)} := \prod_{i=0}^{n-1} \prod_{j=0}^{\ell-1} B_{i,j,d}^{-\tilde{y}^{(b)}_{i \cdot \ell + j, 1}}$
  12. $F_d^{(b)} := e(B_d^{(b)}, L) \cdot k_d^{(b)}$
  13. $\tilde{\tau}_d^{(b)} \leftarrow \mathsf{ZK\text{-}PoK.Audit}(b, \varphi_d^{(b)}, E^{(b)}, k^{(b)})$
  14. $F_m^{(b)} := \prod_{d \in \mathbb{Z}_m} (F_d^{(b)})^{\Delta_{d+1,m}(0)}$
  15. **if** $b = 0$:
  16. $\tau_4^{(b)} := H(e(E^{(b)}, K) / (F_m^{(b)} \cdot C^{(b)}))$
  17. **else**
  18. $\tau_4^{(b)} := H((F_m^{(b)} \cdot C^{(b)}) / e(E^{(b)}, K))$
  19. **return** $\tau^{(b)} := (\{\tau_i^{(b)}\}_{i \in [0,4]}, \{\tilde{\tau}_i^{(b)}\}_{i \in \mathbb{Z}_m})$

- $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$:
  1. **return** $\forall i \in [0,1], \mathsf{VDPF.Verify}(\tau_i^{(0)}, \tau_i^{(1)})$
     $\wedge \; \forall i \in [2,4], \tau_i^{(0)} = \tau_i^{(1)}$
     $\wedge \; \forall i \in \mathbb{Z}_m, \mathsf{ZK\text{-}PoK.Verify}(\tilde{\tau}_i^{(0)}, \tilde{\tau}_i^{(1)})$

Figure 3: The construction of Attr-PC using ZK-PoK, where $f^{(b)}_{\alpha,\beta}$ is sampled from $\mathsf{VDPF.Gen}(1^\lambda, f_{\alpha,\beta})$.

from sk, and the multiplicative shares of $\langle k_i \rangle := e(E_{\alpha,\rho}, A_i)$ with the corresponding ZK-PoK proof for $i \in \mathbb{Z}_m$. According to the SKGen, the elements L, K contain no information about prover's attributes.

Upon receiving the shares of a point function $[\![f_{\alpha,\beta}]\!]$ and the corresponding proof $[\![\pi]\!]$, the verifiers, say P_0 and P_1, locally execute Audit to select the target verification key $\langle \mathsf{vk}_{\alpha,\rho} \rangle$ and obtain the audit tokens. More specifically, denote $[\![\tau]\!] := \{\tau^{(0)}, \tau^{(1)}\}$ as the audit tokens

computed by P_0 and P_1, respectively. The verifiers computes inner product in the exponent between Λ and the evaluation result of $[\![f_{\eta,(\beta,1)}]\!]$ to get $\langle \mathsf{vk}_{\alpha,\rho} \rangle$. Later, they try to recover $\langle C_{\alpha,\rho} \rangle$ using proof shares. For $d \in \mathbb{Z}_m$, suppose $\langle k_d \rangle$ in the proof $[\![\pi]\!] = e(E_{\alpha,\rho}, A_d)$ and A_d corresponds to the same attribute a_d as $B_{\alpha,\rho,d}$. The random mask $H'(a_d) \cdot s_{\alpha,\rho}$ in $B_{\alpha,\rho,d}$ is eliminated by computing $\langle F_d \rangle := e(\langle B_d \rangle, L) \cdot \langle k_d \rangle := e(g,g)^{tr \cdot q(d+1)}$. Therefore, if the proof $[\![\pi]\!]$ is valid, the verifiers can obtain $\langle F_m \rangle := e(g,g)^{tr \cdot q(0)}$ through Lagrange interpolating formula, and $e(E_{\alpha,\rho}, K)/F_m = e(g,g)^{ts_{\alpha,\rho}} = C_{\alpha,\rho}$. The verifiers generate $[\![\tau_4]\!]$ to check this equality. In particular, this process applies prover-assisted bilinear map computation, so that it results a series of ZK-PoK audits and the corresponding tokens $\{[\![\tilde{\tau}_d]\!]\}_{d \in \mathbb{Z}_m}$. Moreover, we introduce $[\![\tau_0]\!], \ldots, [\![\tau_3]\!]$ to ensure the selected verification key belongs to Λ_α. The first two tokens $[\![\tau_0]\!], [\![\tau_1]\!]$ are VDPF tokens of $[\![f_{\alpha,\beta}]\!], [\![f_{\eta,(\beta,1)}]\!]$, respectively. They attest the well-formedness of point functions. The token $[\![\tau_2]\!]$ is computed by hashing the subtractive shares of the different between the sum of the auxiliary outputs of $[\![f_{\eta,(\beta,1)}]\!]$ and the value 1. The equality check of $\tau_2^{(0)} = \tau_2^{(1)}$ in Verify restricts that the special auxiliary output of $[\![f_{\eta,(\beta,1)}]\!]$ is equal to 1. The token $[\![\tau_3]\!]$ is computed by XOR-accumulating the hash values of the subtractive shares of the different $f_{\alpha,\beta}(i) - \sum_{j \in \mathbb{Z}_\ell} \tilde{y}_{i\ell+j,0}$ for $i \in \mathbb{Z}_N, j \in \mathbb{Z}_\ell$, where $\tilde{y}_{i\ell+j,0}$ is the first item of the evaluation result of $f_{\eta,(\beta,1)}(i\ell + j)$. We observe that, the equality $\tau_3^{(0)} = \tau_3^{(1)}$ holds only if $\eta \in [\alpha\ell, \alpha\ell + \ell)$. Finally, the verifiers exchange their audit tokens and then use Verify to determine whether the shared function $[\![f_{\alpha,\beta}]\!]$ should be evaluated or rejected.

Security. We show the security of our Attr-PC scheme described in Fig. 3 with the following theorem, and its proof can be found in Appendix B.2.

Theorem 2. *Let p be a prime chosen with respect to the security parameter λ and let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$ be a bilinear map, where $\mathbb{G}, \mathbb{G}_t$ are two multiplicative cyclic groups with order p. Let $H : \{0,1\}^* \to \{0,1\}^{4\lambda}$ be a hash function sampled from a family $\mathcal{H}$ that is both collision-resistant and XOR-collision-resistant. The Attr-PC construction in Fig. 3 is a secure PC scheme as described in Definition 2 under the decisional BDH assumption.*

5.2 Template-based policy constraints

In this section, we propose a template-based PC (Tpl-PC) scheme to restrict the function output. It is aimed at the point function family $\mathcal{F}$ whose range is a binary field. In our Tpl-PC scheme, each authorized set $\mathcal{Q}_i$ is represented by ℓ restraint strings $\{\mathsf{rs}_{i,j}\}_{j \in \mathbb{Z}_\ell}$, such that $\mathcal{Q}_i := \{f_{i,v} \in \mathcal{F} | \exists j \in \mathbb{Z}_\ell, v \wedge \mathsf{rs}_{i,j} = 0\}$. That is, the restraint string $\mathsf{rs}_{i,j}$ divides the function output into free bits and restricted bits. 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 $R(\cdot,\cdot) \equiv 1$ and omit vk. Note that, it can be extended to fain-grained content constraints by setting different restraint strings according to different verification keys for a non-trivial R.

Fig. 4 presents our Tpl-PC construction. Given the function family $\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{F}_{2^t}\}$, the initialization method $\mathsf{PolicyGen}$ sets up N constraints by determining the restraint strings $\mathsf{rs}_{i,j}$. In Prove, the prover generates a proof for the point function $f_{\alpha,\beta} \in \mathcal{F}$ with the secret key $\emptyset$. The target $\mathsf{rs}_{\alpha,\rho}$ is selected by the condition $\beta \wedge \mathsf{rs}_{\alpha,\rho} = 0$, and the proof shares are a pair of VDPF keys $[\![f_{\eta,\beta}]\!]$ where $\eta := \alpha\ell + \rho$. During the verification process, the verifiers first executes the algorithm Audit to generate audit tokens. More specifically, they obliviously obtain the secret-shared $[\![\mathsf{rs}_{\alpha,\rho} \wedge \beta]\!]$ by computing the inner product between the evaluation result of $[\![f_{\eta,\beta}]\!]$ and the constraints Λ. For $b \in \{0,1\}$, the

Construction Tpl-PC

- **Parameters**: Let $\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{F}_{2^t}\}$ and $\mathcal{F}' := \{f : \{0,1\}^{n+\log \ell} \to \mathbb{F}_{2^t}\}$ be function families of point functions, $N := 2^n$. Let $H : \{0,1\}^* \to \{0,1\}^{4\lambda}$ be a hash function sampled from a family $\mathcal{H}$ that is both collision-resistant and XOR-collision-resistant.
- The SKGen algorithm always outputs $\emptyset$.
- $\mathsf{PolicyGen}(1^\lambda, \mathcal{F})$:
  1. **for** $i \in \mathbb{Z}_N, j \in \mathbb{Z}_\ell$
  2. Choose the string $\mathsf{rs}_{i,j} \in \mathbb{Z}_{2^t}$
  3. $\Lambda_i := \{\mathsf{rs}_{i,j}\}_{j \in \mathbb{Z}_\ell}, \forall i \in \mathbb{Z}_N$
  4. **return** $(\Lambda := \{\Lambda_i\}_{i \in \mathbb{Z}_N}, \emptyset)$
- $\mathsf{Prove}(\Lambda, f_{\alpha,\beta} \in \mathcal{F}, \mathsf{sk} := \emptyset)$:
  1. Select the target $\mathsf{rs}_{\alpha,\rho}$ from Λ, s.t., $\beta \wedge \mathsf{rs}_{\alpha,\rho} = 0$
  2. $\eta := \alpha \cdot \ell + \rho$
  3. $[\![f_{\eta,\beta}]\!] \leftarrow \mathsf{VDPF.Gen}(1^\lambda, f_{\eta,\beta} \in \mathcal{F}')$
  4. $\pi^{(0)} := f^{(0)}_{\eta,\beta}$, $\pi^{(1)} := f^{(1)}_{\eta,\beta}$
  5. **return** $(\pi^{(0)}, \pi^{(1)})$
- $\mathsf{Audit}(b, \Lambda, f^{(b)}_{\alpha,\beta}, \pi^{(b)})$:
  1. Parse $\pi^{(b)} := f^{(b)}_{\eta,\beta}$
  2. $(\{y^{(b)}_i\}, \tau^{(b)}_0) \leftarrow \mathsf{VDPF.Eval}(b, f^{(b)}_{\alpha,\beta}, \mathbb{Z}_N)$
  3. $(\{\widetilde{y}^{(b)}_i\}, \tau^{(b)}_1) \leftarrow \mathsf{VDPF.Eval}(b, f^{(b)}_{\eta,\beta}, \mathbb{Z}_{N\ell})$
  4. $\tau^{(b)}_2 := \bigoplus_{i \in \mathbb{Z}_N} H(y^{(b)}_i \oplus (\bigoplus_{j \in \mathbb{Z}_\ell} \widetilde{y}^{(b)}_{i \cdot \ell + j}) || i)$
  5. $\tau^{(b)}_3 := H(\bigoplus_{i \in \mathbb{Z}_N, j \in \mathbb{Z}_\ell} \mathsf{rs}_{i,j} \wedge \widetilde{y}^{(b)}_{i \cdot \ell + j})$
  6. **return** $\tau^{(b)} := \{\tau^{(b)}_i\}_{i \in [0,3]}$
- $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)})$:
  1. **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 4: The construction of Tpl-PC, where $f^{(b)}_{\alpha,\beta}$ is sampled from $\mathsf{VDPF.Gen}(1^\lambda, f_{\alpha,\beta})$.

verier P_b compute audit token $\tau^{(b)}_3$ as the hash value of the share $(\mathsf{rs}_{\alpha,\rho} \wedge \beta)^{(b)}$. In Verify, the equality check of $\tau^{(0)}_3 = \tau^{(1)}_3$ guarantees $\mathsf{rs}_{\alpha,\rho} \wedge \beta = 0$. To prevent the malicious prover from providing a mismatched $f_{\eta',\beta'}$ to cheat the verifiers, we introduce the additional audit tokens $[\![\tau_0]\!], [\![\tau_1]\!], [\![\tau_2]\!]$ that are similar to our Attr-PC scheme.

Security. We show the security of our Tpl-PC scheme described in Fig. 4 with the following theorem, and its proof can be found in Appendix B.3.

Theorem 3. *Let λ be security parameter. Let $H : \{0,1\}^* \to \{0,1\}^{4\lambda}$ be a hash function sampled from a family $\mathcal{H}$ that is both collision-resistant and XOR-collision-resistant. The Tpl-PC construction in Fig. 4 is a secure PC scheme as described in Definition 2.*

5.3 Efficiency

In this section, we offer an optimization for PC schemes, and discuss the composition of PCs 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 $O(\lambda \cdot (n + \log \ell))$ communication overhead from the prover to the verifiers. As observed in prior work [13], 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.* [14]. Loosely speaking, the 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 constraints. We observe that, for the same function family $\mathcal{F}$, different PC schemes share a common "backbone" of selecting VDPFs. The difference between their selecting VDPFs is the special auxiliary outputs with respect to the different policy

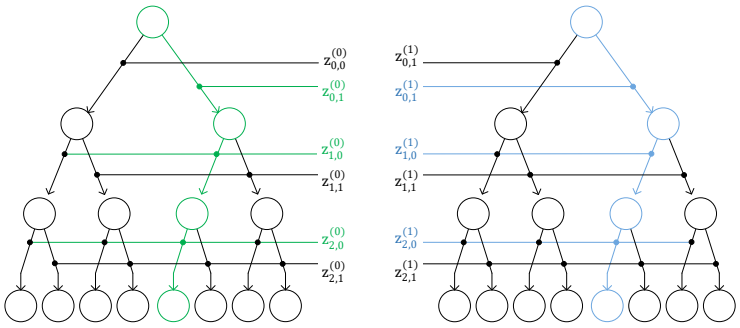

Figure 5: DPF with layer outputs

constraints. Due to the privacy of VDPF, all items in the VDPF evaluation result are pseudo-random and independent to each other. Therefore, multiple PC 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 PCs. The security of the composite construction depends on the security of VDPF and the original multiple PC schemes.

6 PC for DPF with Logarithmic Storage Size

Previous PC schemes (including PACLs in [13]) require that each verifier stores at least N verification keys for N constraints. This is a consequence of the fact that the underlying DPF extends to a vector of length N to indicate the 1-out-of-N constraint selection. This selection step becomes the performance bottleneck when N is large. In this section, we describe how to construct PC with logarithmic storage size for constraint representation to reduce the selection cost.

For simplicity, we focus on $\ell = 1$. Intuitively, we use $2 \log N$ public keys to represent N verification keys as constraints. 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 denote the α-th constraint by $\mathsf{vk}_\alpha := \prod_{i \in [\log N]} k_{i,\alpha_i}$. Only the prover who has the discrete-logarithm knowledge of vk_α can produce a valid proof for α-th constraint.

There are two challenges to this intuition. First, for a point function $f_{\alpha,\beta}$, the verifiers can not obliviously bit-decompose the special input α using standard DPF technique. To overcome this problem, we present a verifiable DPF scheme with layer outputs, called incremental VDPF (IVDPF). In IVDPF, if the evaluation input set contains α, the layer outputs of the evaluation result form the (scaled) bit decomposition of α. The second is that if the secret key is exactly the discrete logarithm of the corresponding verification key, then a malicious user with access to multiple constraints may be able to compute the valid secret keys of other constraints. We introduce bilinear map to address this problem, which blinds the discrete logarithm.

6.1 Incremental VDPF

An incremental 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 $f_{\alpha,\beta}$ with the special

Algorithm $\mathsf{Conv}_{\mathbb{G}}$

- **Parameters**: Let u be the bit length of the element in $\mathbb{G}$. Let $G : \{0,1\}^{\lambda} \to \{0,1\}^{\lambda+u}$ be a PRG.
- $\mathsf{Conv}_{\mathbb{G}}(s)$:

1 $v := G(s)$
2 **if** $|\mathbb{G}| = 2^u$: **return** the group element represented by the first u bits of v
3 **return** the group element corresponding to v

Figure 6: Pseudocode for converting a string s to an element in $\mathbb{G}$. Due to the secure PRG G, if the string s is random, $\mathsf{Conv}_{\mathbb{G}}(s)$ is a pseudo-random element in $\mathbb{G}$.

layer outputs $\{\beta'_i\}$. 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 the layer outputs with a token. The verification function Verify takes input as tokens, and outputs 1 for accept if and only if the function outputs have at most one non-zero item and the layer outputs match with the special input α. The formal definition of our IVDPF is described in Definition 4, which can be found in Appendix A.

Our IVDPF construction is inspired by the CPRG optimization of distributed DPF generation [3] and the incremental DPF [6]. Recap the underlying DPF representation [14] of the VDPF solution [21]. For a point function $f_{\alpha,\beta} : \{0,1\}^n \to \mathbb{G}$, each of the two DPF keys defines a GGM-style binary tree [26] with 2^n leaves, and every node in the tree is labeled by a pseudo-random value. The DPF construction guarantees the invariant of a pair of keys at each level i. That is, if a node is in the special path (i.e., the path from the root to the α-th leaf), its labels in two trees are different and independent; otherwise, its labels are identical. Let α_i denote the i-th bit of α. As shown in Fig. 5, for the accumulations $z^{(b)}_{i,0}, z^{(b)}_{i,1}$ of all left children and right children at level i, we have $z^{(0)}_{i,\alpha_i} \neq z^{(1)}_{i,\alpha_i}$ and $z^{(0)}_{i,\bar{\alpha}_i} = z^{(1)}_{i,\bar{\alpha}_i}$. Therefore, we can instruct the i-th layer output by making $z^{(0)}_{i,\alpha_i}, z^{(1)}_{i,\alpha_i}$ reconstruct β'_i and correcting $z^{(0)}_{i,\bar{\alpha}_i}, z^{(1)}_{i,\bar{\alpha}_i}$ to 0.

The details of our IVDPF is illustrated in Fig. 7. In particular, the label of each note 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 by [6], 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 of Conv is employed to expend $\tilde{s}^{(b)}$ to a new pseudo-random seed $s^{(b)}$ for children and an element $w^{(b)}$ for the layer output.

Preventing malicious dealer. The VDPF technique from [21] 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 [21] to check each level. During the key generation, for each level i, we introduce a verification correction word cs_i to correct the hash values of the different

Construction IVDPF

- **Parameters**: Let $\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{G}\}$ be a function family of point functions, and $\mathbb{G}'_1, \ldots, \mathbb{G}'_n$ be layer output groups. Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ and $\mathsf{Conv}_\mathbb{G} : \{0,1\}^n \to \mathbb{G}$ (cf. Fig. 6) be two 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 both 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: $\mathsf{keep} := R, \mathsf{lose} := L$
  $s^{\mathsf{cw}} := s_{\mathsf{lose}}^{(0)} \oplus s_{\mathsf{lose}}^{(1)}$
  $t_L^{\mathsf{cw}} := t_L^{(0)} \oplus t_L^{(1)} \oplus \alpha_i \oplus 1$
  $t_R^{\mathsf{cw}} := t_R^{(0)} \oplus t_R^{(1)} \oplus \alpha_i$
  $\mathsf{cw}_i := (s^{\mathsf{cw}}, t_L^{\mathsf{cw}}, t_R^{\mathsf{cw}})$
  $\tilde{s}_i^{(b)} := s_{\mathsf{keep}}^{(b)} \oplus t_{i-1}^{(b)} \cdot s^{\mathsf{cw}}$ for $b \in \{0,1\}$
  $t_i^{(b)} := t_{\mathsf{keep}}^{(b)} \oplus t_{i-1}^{(b)} \cdot t_{\mathsf{keep}}^{\mathsf{cw}}$ for $b \in \{0,1\}$
  $\alpha_{[1,i]} := \alpha_1||\ldots||\alpha_i$
  $\mathsf{cs}_i := H(\tilde{s}_i^{(0)}||t_i^{(0)}||\alpha) \oplus H(\tilde{s}_i^{(1)}||t_i^{(1)}||\alpha_{[1,i]})$
  $s_i^{(b)}||w_i^{(b)} \leftarrow \mathsf{Conv}_{\{0,1\}^\lambda \times \mathbb{G}'_i}(\tilde{s}_i^{(b)})$
  $\mathsf{lcw}_i := (-1)^{t_i^{(1)}} \cdot [\beta'_i - w_i^{(0)} + w_i^{(1)}]$
$\mathsf{ocw} := (-1)^{t_n^{(1)}} \cdot (\beta - \mathsf{Conv}_\mathbb{G}(s_n^{(0)}) + \mathsf{Conv}_\mathbb{G}(s_n^{(1)}))$
$f_{\alpha,\beta}^{(0)} := (s_0^{(0)}, (\mathsf{cw}_i, \mathsf{cs}_i, \mathsf{lcw}_i)_{i\in[n]}, \mathsf{ocw})$
$f_{\alpha,\beta}^{(1)} := (s_0^{(1)}, (\mathsf{cw}_i, \mathsf{cs}_i, \mathsf{lcw}_i)_{i\in[n]}, \mathsf{ocw})$
return $(f_{\alpha,\beta}^{(0)}, f_{\alpha,\beta}^{(1)})$

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

Parse $f_{\alpha,\beta}^{(b)} := (s_0, t_0, (\mathsf{cw}_i, \mathsf{lcw}_i)_{i\in[n]}, \mathsf{ocw})$
$t_0 := b, \tau_i := \mathsf{cs}_i$
$z_i^{(b)} := 0$ for $i \in [n]$
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: $\mathsf{keep} := R$
    $\tilde{s}_i||t_i := (s_{\mathsf{keep}}||t_{\mathsf{keep}}) \oplus t_{i-1} \cdot (s^{\mathsf{cw}}||t_{\mathsf{keep}}^{\mathsf{cw}})$
    $x_{[1,i]} := x_1||\ldots||x_i$
    $\tau_i := \tau_i \oplus H'(\tau_i \oplus H(\tilde{s}_i||t_i||x_{[1,i]}) \oplus t_i \cdot \mathsf{cs}_i)$
    $s_i||w_i \leftarrow \mathsf{Conv}_{\{0,1\}^\lambda \times \mathbb{G}'_i}(\tilde{s}_i)$
    $z_{i,x_i}^{(b)} := z_{i,x_i}^{(b)} + (-1)^b \cdot (w_i + t_i \cdot \mathsf{lcw}_i)$
$y_x^{(b)} := (-1)^b \cdot (\mathsf{Conv}_\mathbb{G}(s_n) + t_n \cdot \mathsf{ocw})$
$\tau^{(b)} := \bigoplus_{i\in[n]} \tau_i$
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 7: The construction of incremental VDPF.

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. Since a correct word 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 scheme described in Fig. 7 with the following theorem, and its proof can be found in Appendix B.4.

Theorem 4. *Let λ be the security parameter. Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ and $\mathsf{Conv}_\mathbb{G} : \{0,1\}^n \to \mathbb{G}$ be pseudorandom generators, $H : \{0,1\}^{\lambda+1+n} \to \{0,1\}^{4\lambda}$ be a hash function sampled from a family $\mathcal{H}$ that is both 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. The construction from 7 is a secure IVDPF scheme for the function family $\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{G}\}$ of point functions and the layer output group $\{\mathbb{G}'_i\}_{i\in[n]}$, as described in Definition 4.*

Construction IVDPF-PC

- **Parameters**: Let $\mathcal{F} := \{f : \{0,1\}^n \to \tilde{\mathbb{G}}\}$ be function family of point functions, and $N := 2^n$. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$ be a bilinear map, where $\mathbb{G}, \mathbb{G}_t$ are two multiplicative cyclic groups with the prime order p, and g is the generator of $\mathbb{G}$. Let $H : \{0,1\}^* \to \{0,1\}^{2\lambda}$ be a hash function sampled from a family $\mathcal{H}$ that is collision-resistant.

- $\mathsf{PolicyGen}(1^\lambda, \mathcal{F})$:

for $i \in [n]$:
  $r_{i,0}, r_{i,1} \leftarrow \mathbb{Z}_p$
  $k_{i,0} := e(g,g)^{r_{i,0}}, k_{i,1} := e(g,g)^{r_{i,1}}$
$\Lambda := \{(k_{i,0}, k_{i,1})\}_{i\in[n]}$, $\xi := \{r_{i,0}, r_{i,1}\}_{i\in[n]}$
return (Λ, ξ)

- $\mathsf{SKGen}(1^\lambda, \alpha, \xi)$:

$d := \sum_{i\in[n]} r_{i,\alpha_i}$
$c \leftarrow \mathbb{Z}_p^*$, $\mathsf{sk}_0 := g^c, \mathsf{sk}_1 := d/c$
return $\mathsf{sk} := (\mathsf{sk}_0, \mathsf{sk}_1)$

- $\mathsf{Prove}(\Lambda, f_{\alpha,\beta} \in \mathcal{F}, \mathsf{sk})$:

Parse $\mathsf{sk} := (\mathsf{sk}_0, \mathsf{sk}_1)$
$s \leftarrow \mathbb{Z}_p^*, u := \mathsf{sk}_0^s$
$[\![v]\!] \leftarrow \mathsf{AddShare}_{\mathbb{Z}_p,2}(\mathsf{sk}_1/s)$
$\pi^{(0)} := (u, v^{(0)})$, $\pi^{(1)} := (u, v^{(1)})$
return $(\pi^{(0)}, \pi^{(1)})$

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

Parse $\pi^{(b)} := (u, v^{(b)})$
Parse $\Lambda := \{(k_{i,0}, k_{i,1})\}_{i\in[n]}$
$(Y, Z, \tau_0^{(b)}) \leftarrow \mathsf{IVDPF.Eval}(b, f^{(b)}_{\alpha,\beta}, \mathbb{Z}_N)$
Parse $Z := \{z^{(b)}_{i,0}, z^{(b)}_{i,1}\}_{i\in[n]}$
$\tau_1^{(b)} := \bigoplus_{i\in[n]} H(z^{(b)}_{i,0} + z^{(b)}_{i,1} - b)$
$\mathsf{vk}^{(b)} := \prod_{i\in[n]} k_{i,0}^{z^{(b)}_{i,0}} \cdot k_{i,1}^{z^{(b)}_{i,1}}$
if $b = 0$:
  $\tau_2^{(b)} := H(e(u, g^{v^{(b)}})/\mathsf{vk}^{(b)})$
else:
  $\tau_2^{(b)} := H(\mathsf{vk}^{(b)}/e(u, g^{v^{(b)}}))$
return $\tau^{(b)} := (\tau_0^{(b)}, \tau_1^{(0)}, \tau_2^{(0)})$

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

return $\mathsf{IVDPF.Verify}(\tau_0^{(0)}, \tau_0^{(1)}) \wedge \forall i \in [2], \tau_i^{(0)} = \tau_i^{(1)}$

Figure 8: The construction of IVDPF-PC, where $f^{(b)}_{\alpha,\beta}$ is sampled by $\mathsf{IVDPF.Gen}(1^\lambda, f_{\alpha,\beta}, \{1 \in \mathbb{Z}_p\}_n)$.

6.2 Construction using incremental VDPF

We describe our PC scheme from incremental VDPF in Fig. 8. It is aimed to constraint the function family $\mathcal{F} := \{f : \{0,1\}^n \to \tilde{\mathbb{G}}\}$ of point functions. Let $N := 2^n$. Our IVDPF-PC scheme supports N different constraints for $\mathcal{F}$ and only require each verifier stores $2 \log N$ public keys $\{k_{i,0}, k_{i,1}\}_{i\in[n]}$. Our intuition is that verifiers obliviously obtain the target 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 of the shared function $f_{\alpha,\beta}$, and then check that the prover holds a valid secret key encoding the discrete logarithm of the corresponding vk_α.

To defend against the attack on the linear composition of discrete logarithms, we generate the verification keys using a biliear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$. Let g be the generator of $\mathbb{G}$. Each verifier stores $(e(g,g)^{r_{i,0}}, e(g,g)^{r_{i,1}})_{i\in[n]}$ to represent Λ. The algorithm SKGen products the randomized secret key use the trapdoor $\xi := \{r_{i,0}, r_{i,1}\}_{i\in[n]}$. More specifically, for each $\mathsf{vk}_\alpha \in \Lambda$, $d := \sum_{i\in[n]} r_{i,\alpha_i}$ is the discrete logarithm of vk_α to the base $e(g,g)$. SKGen samples random $c \in \mathbb{Z}_p$ and outputs a secret key as $\mathsf{sk} := (g^c, d/c)$. Note that, $e(g^c, g^{d/c}) = e(g,g)^{c \cdot d/c} = e(g,g)^d$. The proof generation method Prove re-randomizes two items in sk to obtain u, v respectively. It then produces the proof shares consisting of the random u and the additive shares of $[\![v]\!]$. Upon receiving the proof shares, the verifiers can jointly compute $\langle e(u, g^v)/\mathsf{vk}_\alpha \rangle$ to obtain the audit tokens $\tau_2^{(0)}, \tau_2^{(1)}$ using Audit, and then exchange tokens to check $e(u, g^v)/\mathsf{vk}_\alpha = 1$ in Verify. In addition, we let the verifiers verify the well-formedness of the function shares using IVDPF scheme and check whether each special layer output is equal to 1. It guarantees that the verifiers obtain a valid verification

Table 1: Performance of proof generation. For our Attr-PC scheme, we set each constraint to 5-DNF.

# of keys per Λ_i	1	2^5	2^{10}	2^{15}
DPF-PACL [13]	0.3 μs	1.37 μs	2.74 μs	3.13 μs
VDPF-PACL [13]	30.1 ms	29.3 ms	29.9 ms	29.3 ms
Attr-PC (Ours)	23.1 ms	23.1 ms	23.2 ms	22.8 ms
Tpl-PC (Ours)	0.6 μs	1.49 μs	2.65 μs	3.05 μs
IVDPF-PC (Ours)	31.7 μs	41.6 μs	50.2 μs	63.8 μs
SK-Tpl-PC	29.1 ms	30.2 ms	29.4 ms	29.3 ms
Attr-Tpl-PC	23.0 ms	23.6 ms	23.2 ms	22.8 ms

key.

Security. We show the security of our IVDPF-PC scheme described in Fig. 8 with the following theorem, and its proof can be found in Appendix B.5.

Theorem 5. *Let p be a prime chosen with respect to the security parameter λ. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$ be a bilinear map, where $\mathbb{G}, \mathbb{G}_t$ are two multiplicative cyclic groups with order p. Let $H : \{0,1\}^* \to \{0,1\}^{4\lambda}$ be a hash function sampled from a family $\mathcal{H}$ that is both collision-resistant and XOR-collision-resistant. The IVDPF-PC construction in Fig. 8 is a secure PC scheme as described in Definition 2 under the decisional BDH assumption.*

7 Implementation and benchmark

Our PC schemes are implemented in Go and C. We implement the underlying DPF followed by [14, 21] and employ the optimization of the FSS tensor technique in [14]. 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 [27], and adopt the efficient BLS12-381 implementation in [28]. 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 [13]. We rerun the source code [29] 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 [30]. In addition, we evaluate two PC schemes for multi-policy: the SK-Tpl-PC composed of the VDPF-PACL and our Tpl-PC; and the Attr-Tpl-PC composed of our Attr-PC and Tpl-PC.

Parameters. Set $\lambda = 128$ and $n = 32$. In practical, such as mailbox system [10], constraints are applied to a registered set X, and the verifiers evaluate DPF on X rather than full domain. Denote $N = |X|$ as the number of constraints, ℓ as the number of verification keys per constraint.

Optimizations. We reduce the communication cost between the verifiers by XOR-accumulating some audit tokens (including the VDPF/IVDPF tokens), which are hash values and verified by a simple equality check in Verify.

Table 2: Proof size and audit token size in bytes, where s_ℓ is the standard DPF key size for the domain size ℓ and m is the conjunction size in our Attr-PC scheme.

	Proof Size	Token Size
DPF-PACL [13]	$32 + s_\ell$	64
VDPF-PACL [13]	$1952 + s_\ell + 64$	816
Fixed VDPF-PACL	$1952 + s_\ell + 64$	880
Attr-PC (Ours)	$584m + s_\ell + 64$	$64 + 1240m$
Tpl-PC (Ours)	$s_\ell + 64$	64
IVDPF-PC (Ours)	$128 + s_\ell + 96 \log \ell$	64
SK-Tpl-PC	$1952 + s_\ell + 64$	880
Attr-Tpl-PC	$584m + s_\ell + 64$	$64 + 1240m$

7.1 Prover costs

Table 1 compares the proving costs of schemes. We note that, using the FSS tensor optimization, the computational complexity of Prove is log-linear in ℓ. The DPF-PACL and our Tpl-PC schemes achieve an overwhelming performance advantage, because their operations are minimal except for generating (V)DPF keys. Our IVDPF-PC requires $O(n)$ group operations for layer outputs in IVDPF key generation, resulting in additional running time. In our Attr-PC scheme, the prover needs to generate the ZK-PoK proof for each attribute, which is bottlenecked by group exponentiation operations. However, the proving time of our Attr-PC remains below 25 ms for $\ell \leq 2^{15}$. For the composite PC schemes, multiple policies 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.

7.2 Communication costs

There are two communication rounds in our PC schemes and PACLs: one is that the prover distributes the proof shares to the verifiers; the other is that two verifiers exchange their audit tokens. Table 2 reports the concrete communication costs 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 [21] is $s_\ell + 64$ bytes, and our IVDPF key size is $s_\ell + 96 \log \ell$ bytes due to the $\log \ell$ correction seeds and layer output correction words.

The proof size and token size of our Tpl-PC and IVDPF-PC are smaller than those of VDPF-PACL. For our Attr-PC scheme, its proof size is linear in the size m of each conjunction, where m is the number of required attributes per verification. Since the VDPF-PACL scheme supports the verification with only one "attribute", our Attr-PC scheme outperforms the VDPF-PACL for the amortized proof size over the number of attributes. Compared to the sum of multiple PCs, the composite PC schemes save 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 [13]. As shown in Table 2, our security improvement is almost free, in fact it only increases the token size by 64 bytes.

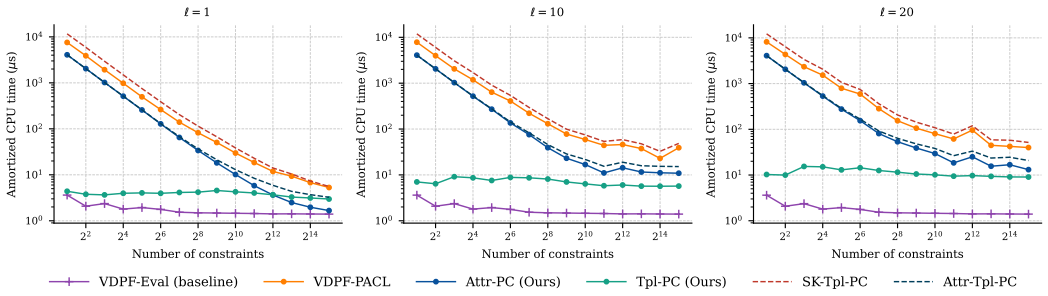

Figure 9: Comparison of the amortized verification overhead, where VDPF-PACL refers to [13]. The run-time is amortized by the number of constraints. Our Attr-PC scheme is additionally amortized by the conjunction size $m = 5$.

7.3 Verification costs

The verifiers executes Audit and Verify sequentially for verification. Thus, we take the total run-time of the two algorithm as the verification cost. We compare our Attr-PC for 5-DNF, Tpl-PC and the VDPF-PACL schemes in Fig. 9, while the VDPF evaluation of $f_{\alpha,\beta}$ is the baseline. As the number of constraints increases, the amortized overhead of VDPF-PACL and Attr-PC decreases. This is because the overhead of the SPoSS/ZK-PoK and the equality check are amortized over the number of constraints. On the contrary, the amortized overhead of our Tpl-PC plateaus at the beginning due to its minimal operations.

Furthermore, with the amortization of the attribute number, the verification time of our Attr-PC scheme is 2× - 4× faster than the VDPF-PACL as shown in Fig. 9. To demonstrate the performance of our Attr-PC scheme with respect to the number of attributes, we report its running time amortized only the constraint number in Fig. 10. We observe that the verification cost of Attr-PC is linear in the number of attributes, while the gradient decreases as the number of constraints grows. For the composite PCs, their verification time are less than the sum of the corresponding single-policy schemes because it halves the VDPF evaluations. This advantage would be significant when the domain of the selecting point function is large.

To demonstrate the advantage of our IVDPF-PC scheme, we plot the run-time of ver-

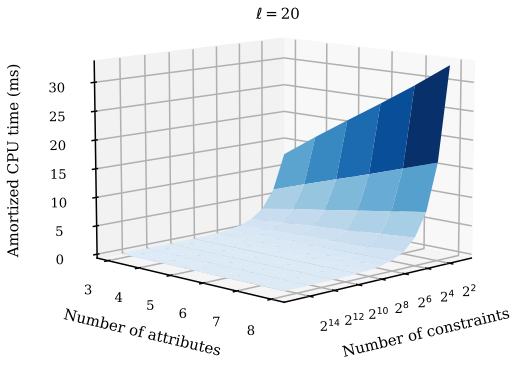

Figure 10: Verification time of our Attr-PC amortized only by the number of constraints.

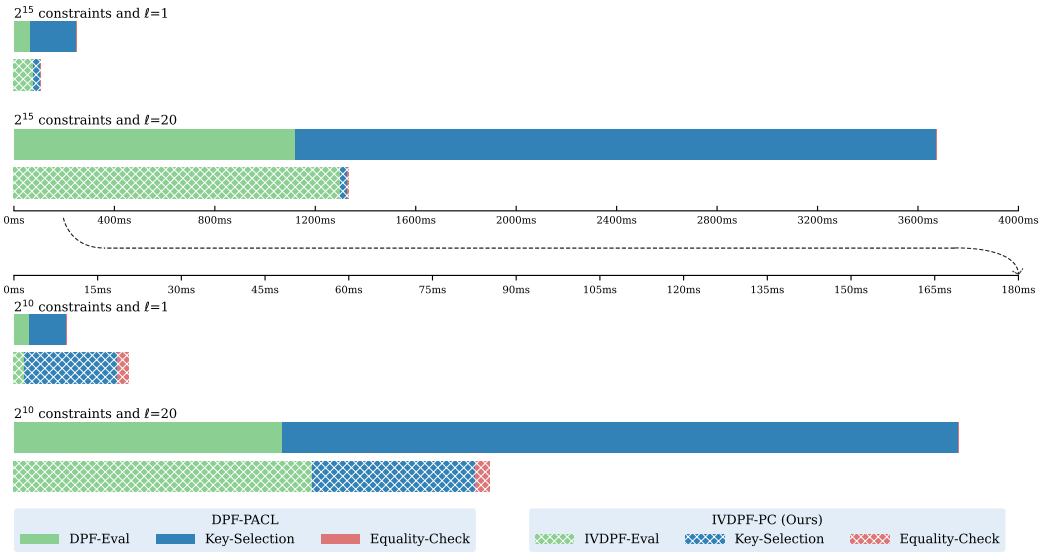

Figure 11: Verification performance of DPF-PACL [13] and our IVDPF-PC scheme.

ification by steps in Fig. 11. It shows that, for the DPF-PACL scheme of [13], obliviously selecting the target verification key is the most costly step. Because the DPF-PACL uses N verification keys to represent N constraint, and requires $O(N\ell)$ group exponentiation operations in the key selection step. Our IVDPF-PC scheme reduce the representation of the constraint list to $2\log N$ public keys, so that only $O(\ell \log N)$ group exponentiation operations are required. Therefore, our IVDPF-PC scheme is significantly more efficient than the DPF-PACL when N and ℓ are large. For instance, our IVDPF-PC scheme improves performance by a factor of $2.5\times$ with 2^{15} constraints.

References

[1] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *EUROCRYPT 2015, Part II*, ser. LNCS, E. Oswald and M. Fischlin, Eds., vol. 9057. Sofia, Bulgaria: Springer, Heidelberg, Germany, Apr. 26–30, 2015, pp. 337–367.

[2] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *EUROCRYPT 2014*, ser. LNCS, P. Q. Nguyen and E. Oswald, Eds., vol. 8441. Copenhagen, Denmark: Springer, Heidelberg, Germany, May 11–15, 2014, pp. 640–658.

[3] J. Doerner and a. shelat, "Scaling ORAM for secure computation," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. Dallas, TX, USA: ACM Press, Oct. 31 – Nov. 2, 2017, pp. 523–535.

[4] P. Bunn, J. Katz, E. Kushilevitz, and R. Ostrovsky, "Efficient 3-party distributed ORAM," in *SCN 20*, ser. LNCS, C. Galdi and V. Kolesnikov, Eds., vol. 12238. Amalfi, Italy: Springer, Heidelberg, Germany, Sep. 14–16, 2020, pp. 215–232.

[5] K. Ji, B. Zhang, T. Lu, and K. Ren, "Multi-party private function evaluation for ram," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1252–1267, 2023.

[6] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Lightweight techniques for private heavy hitters," in *2021 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 24–27, 2021, pp. 762–776.

[7] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *USENIX Security 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, Aug. 12–14, 2020, pp. 2505–2522.

[8] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function secret sharing for mixed-mode and fixed-point secure computation," in *EUROCRYPT 2021, Part II*, ser. LNCS, A. Canteaut and F.-X. Standaert, Eds., vol. 12697. Zagreb, Croatia: Springer, Heidelberg, Germany, Oct. 17–21, 2021, pp. 871–900.

[9] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. R. Bach, "AriaNN: Low-interaction privacy-preserving deep learning via function secret sharing," *PoPETs*, vol. 2022, no. 1, pp. 291–316, Jan. 2022.

[10] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh, "Express: Lowering the cost of metadata-hiding communication with cryptographic privacy," in *USENIX Security 2021*, M. Bailey and R. Greenstadt, Eds. USENIX Association, Aug. 11–13, 2021, pp. 1775–1792.

[11] A. Vadapalli, K. Storrier, and R. Henry, "Sabre: Sender-anonymous messaging with fast audits," in *2022 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 22–26, 2022, pp. 1953–1970.

[12] Z. Newman, S. Servan-Schreiber, and S. Devadas, "Spectrum: High-bandwidth anonymous broadcast," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*.

[13] S. Servan-Schreiber, S. Beyzerov, E. Yablon, and H. Park, "Private access control for function secret sharing," Cryptology ePrint Archive, Report 2022/1707, 2022, https://eprint.iacr.org/2022/1707.

[14] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *ACM CCS 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. Vienna, Austria: ACM Press, Oct. 24–28, 2016, pp. 1292–1303.

[15] D. Boneh and M. K. Franklin, "Identity-based encryption from the Weil pairing," in *CRYPTO 2001*, ser. LNCS, J. Kilian, Ed., vol. 2139. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 19–23, 2001, pp. 213–229.

[16] A. Sahai and B. R. Waters, "Fuzzy identity-based encryption," in *EUROCRYPT 2005*, ser. LNCS, R. Cramer, Ed., vol. 3494. Aarhus, Denmark: Springer, Heidelberg, Germany, May 22–26, 2005, pp. 457–473.

[17] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *ACM CCS 2006*, A. Juels, R. N. Wright, and S. De Capitani di Vimercati, Eds. Alexandria, Virginia, USA: ACM Press, Oct. 30 – Nov. 3, 2006, pp. 89–98, available as Cryptology ePrint Archive Report 2006/309.

[18] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE Symposium on Security and Privacy*. Oakland, CA, USA: IEEE Computer Society Press, May 20–23, 2007, pp. 321–334.

[19] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An anonymous messaging system handling millions of users," in *2015 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 17–21, 2015, pp. 321–338.

[20] M. N. Wegman and L. Carter, "New hash functions and their use in authentication and set equality," *Journal of Computer and System Sciences*, vol. 22, pp. 265–279, 1981.

[21] L. de Castro and A. Polychroniadou, "Lightweight, maliciously secure verifiable function secret sharing," in *EUROCRYPT 2022, Part I*, ser. LNCS, O. Dunkelman and S. Dziembowski, Eds., vol. 13275. Trondheim, Norway: Springer, Heidelberg, Germany, May 30 – Jun. 3, 2022, pp. 150–179.

[22] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *CRYPTO'91*, ser. LNCS, J. Feigenbaum, Ed., vol. 576. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 11–15, 1992, pp. 420–432.

[23] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *14th USENIX Conference on Networked Systems Design and Implementation (NSDI'17)*.

[24] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO'86*, ser. LNCS, A. M. Odlyzko, Ed., vol. 263. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1987, pp. 186–194.

[25] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Zero-knowledge proofs on secret-shared data via fully linear PCPs," in *CRYPTO 2019, Part III*, ser.

LNCS, A. Boldyreva and D. Micciancio, Eds., vol. 11694. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 18–22, 2019, pp. 67–97.

[26] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions (extended abstract)," in *25th FOCS*. Singer Island, Florida: IEEE Computer Society Press, Oct. 24–26, 1984, pp. 464–479.

[27] P. S. L. M. Barreto, B. Lynn, and M. Scott, "Constructing elliptic curves with prescribed embedding degrees," in *SCN 02*, ser. LNCS, S. Cimato, C. Galdi, and G. Persiano, Eds., vol. 2576. Amalfi, Italy: Springer, Heidelberg, Germany, Sep. 12–13, 2003, pp. 257–267.

[28] S. Kumar, Y. Hu, M. P. Andersen, R. A. Popa, and D. E. Culler, "JEDI: Many-to-many end-to-end encryption and key delegation for IoT," in *USENIX Security 2019*, N. Heninger and P. Traynor, Eds. Santa Clara, CA, USA: USENIX Association, Aug. 14–16, 2019, pp. 1519–1536.

[29] S. Servan-Schreiber, S. Beyzerov, E. Yablon, and H. Park, "PACL source code," github, 2022, https://github.com/sachaservan/pacl.

[30] M. Kojo and T. Kivinen, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)," RFC 3526, 2003. [Online]. Available: https://www.rfc-editor.org/info/rfc3526

A Definition of incremental VDPF

We define the incremental VDPF as follows:

Definition 4 (Incremental VDPF)**.** *Let $\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{G}\}$ be a function family of point functions, and $\{\mathbb{G}'_i\}_{i\in[n]}$ are groups for special layer outputs. A 2-party incremental VDPF scheme for $\mathcal{F}$ and $\{\mathbb{G}'_i\}_{i\in[n]}$ consists of three PPT algorithms* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Verify})$ *defined as follows:*

- $(f^{(0)}_{\alpha,\beta}, f^{(1)}_{\alpha,\beta}) \leftarrow \mathsf{Gen}(1^\lambda, f_{\alpha,\beta}, \{\beta'_i\}_{i\in[n]})$ *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, i.e., the additive shares of $[\![f_{\alpha,\beta}]\!]$.*
- $(\{y^{(b)}_x\}_{x\in X}, \{z_{i,0}, z_{i,1}\}_{i\in[n]}, \tau^{(b)}) \leftarrow \mathsf{Eval}(b, f^{(b)}_{\alpha,\beta}, X)$ *is the verifiable evaluation algorithm that takes input as an index $b \in \{0,1\}$, a 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 the FSS outputs, which 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 that is used to verify the well-formedness of the shared function.*
- $1/0 \leftarrow \mathsf{Verify}(\tau^{(0)}, \tau^{(1)}$ *is the verification algorithm that takes input as two tokens. It outputs 1 for accept or outputs 0 for reject.*

A secure IVDPF must satisfy three properties as follows:

- ***Correctness.*** *For all* $f_{\alpha,\beta} \in \mathcal{F}$, $X \subseteq \{0,1\}^n$, $\{\beta'_i \in \mathbb{G}_i\}_{i\in[n]}$, *let* $(f^{(0)}_{\alpha,\beta}, f^{(1)}_{\alpha,\beta}) \leftarrow \mathsf{Gen}(1^\lambda, f_{\alpha,\beta}, \{\beta_i\}_{i\in[n]})$, *it hold 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, $\alpha_{[1,i]}$ *denotes the* i*-bit prefix of* α, *and* $X_{[1,i]}$ *denotes 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* $\mathsf{View}_{\mathsf{VDPF}}(b, f, B, X)$ *as probability distribution ensemble* $\{(f^{(b)}, \tau^{(1-b)})\}_\lambda$, *where* $(f^{(0)}, f^{(1)})$ *is sampled by* $\mathsf{Gen}(1^\lambda, f, B)$, *and* $(\{y^{(1-b)}_x\}_{x\in X}, \{z_{i,0}, z_{i,1}\}_{i\in[n]}, \tau^{(1-b)})$ *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 following two distributions are computationally indistinguishable:*

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

- ***Soundness.*** *We say IVDPF is computationally sound, if* $\forall X \subseteq \{0,1\}^n$ *and* $(\{y^{(b)}_x\}_{x\in X}, \{z^{(b)}_{i,0}, z^{(b)}_{i,1}\}_{i\in[n]}, \tau^{(b)}) \leftarrow \mathsf{Eval}(b, f^{(b)}_{\alpha,\beta}, X)$ *for* $b \in \{0,1\}$, *the following probability is at least* $1 - \mathsf{negl}(\lambda)$

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

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

B Security proofs

B.1 Proof of Theorem 1

Proof. We prove the ZK-PoK construction in Fig. 2 satisfies the properties in Definition 3.

Completeness. We show that if $k := e(E, k)$, then Verify outputs 1. In Fig. 2, Verify outputs 1 if and only if $\hat{r}^{(0)} \oplus \hat{r}^{(1)} = r$, $T^{(0)}_0 \cdot T^{(1)}_0 = B$, $T^{(0)}_1 \cdot T^{(1)}_1 = C$ and $D^{(0)} \cdot D^{(1)} = 1$. The equality of $\hat{r}^{(0)} \oplus \hat{r}^{(1)} = r$, $T^{(0)}_0 \cdot T^{(1)}_0 = B$, $T^{(0)}_1 \cdot T^{(1)}_1 = C$ follows by inspection. To see why it holds that $D^{(0)} \cdot D^{(1)} = 1$, observe that

$$\begin{aligned} D^{(0)} \cdot D^{(1)} =& e(C, B) \cdot e(C, g)^x \cdot e(g, B)^y \cdot e(g, g)^z \cdot k^{-r} \\ =& e(E \cdot g^{-y}, A^r \cdot g^{-x}) \cdot e(E \cdot g^{-y}, g)^x \end{aligned}$$

$$
\begin{aligned}
&\cdot e(g, A^r \cdot g^{-x})^y \cdot e(g,g)^z \cdot k^{-r} \\
=&e(E,A)^r \cdot e(g,g)^{-xy} \cdot e(g,g)^z \cdot k^{-r} \\
=&e(E,A)^r \cdot k^{-r} \\
=&1 \text{ by assumption that } k = e(E,A)
\end{aligned}
$$

Knowledge soundness. We construct an efficient extractor $\mathcal{E}$ that recovers A with respect to k, E from the output of an adversary $\mathcal{A}$. The extractor $\mathcal{E}$ proceeds as follows:

1 Run $\mathcal{A}(E,k)$ to obtain its output $(\pi^{(0)}, \pi^{(1)})$ where

$$
\begin{aligned}
\pi^{(0)} &:= (A^{(0)}, B, C, x^{(0)}, y^{(0)}, z^{(0)}, r, s^{(0)}) \\
\pi^{(1)} &:= (A^{(1)}, B, C, x^{(1)}, y^{(1)}, z^{(1)}, r, s^{(1)})
\end{aligned}
$$

2 Output $A := A^{(0)} \cdot A^{(1)}$

If $(\pi^{(0)}, \pi^{(1)})$ is a valid ZK-PoK proof, then $T_1^{(0)} T_1^{(1)} = C$ and $D^{(0)} D^{(1)} = 1$. In turn, we have that (1) $E^r \cdot g^{-y'} = (E')^r \cdot g^{-y}$, and (2) $e(E', A)^r \cdot e(g,g)^{-xy'} \cdot e(g,g)^z \cdot k^{-r} = 1$ for some randomness r. The adversary $\mathcal{A}$ can choose arbitrary x, y, y', z, E'. For (1), let $\Delta_y := y - y'$, $\Delta_E := E'/E$. We have $E^r \cdot g^{\Delta_y - y} \cdot (\Delta_E \cdot E)^{-r} \cdot g^y = 1$ which reduces to $g^{\Delta_y} \cdot \Delta_E^{-r} = 1$. Thus, either (a) the adversary $\mathcal{A}$ obtains r from the random oracle H such that $g^{\Delta_y} \cdot \Delta_E^{-r} = 1$, (which happens with negligible probability), or (b) $\Delta_y = 0$ and $\Delta_E = 1$, i.e., $E = E', y = y'$. Therefore, we can assume $E = E', y = y'$ in (2). Suppose that $z = xy + \Delta$ for some Δ [23]. We have $e(E,A)^r \cdot e(g,g)^{-xy} \cdot e(g,g)^z \cdot k^{-r} + e(g,g)^{\Delta}$ which reduces to $(e(E,A)/k)^r \cdot e(g,g)^{\Delta} = 1$. Thus, either (a) the malicious prover obtained r from the random oracle H such that $(e(E,A)/k)^r = e(g,g)^{-\Delta}$, (which happens with negligible probability), or (b) $e(E,A) = k$ and $\Delta = 0$. Therefore, the output A by the extractor $\mathcal{E}$ is the argument of the bilinear map with respect to E, k, as required.

Zero knowledge. We construct an efficient simulator Sim for the view of any verifier P_b. On input $(1^\lambda, b, \mathbb{G}, \mathbb{G}_t)$, the simulator Sim proceeds as follows:

1 $A^{(b)} \leftarrow \mathbb{G}$, $x^{(b)}, y^{(b)}, z^{(b)} \leftarrow \mathbb{Z}_p$;
2 $\hat{r}^{(b)}, r, s^{(b)} \leftarrow \{0,1\}^\lambda$, $B, C \leftarrow \mathbb{G}$;
3 $\pi^{(b)} := (A^{(b)}, B, C, x^{(b)}, y^{(b)}, z^{(b)}, r, s^{(b)})$;
4 $T_0^{(1-b)}, T_1^{(1-b)} \leftarrow \mathbb{G}$, $D^{(1-b)} \leftarrow \mathbb{G}_t$;
5 $\tau^{(1-b)} := (T_0^{(1-b)}, T_1^{(1-b)}, D^{(1-b)}, r \oplus \hat{r}^{(b)}, r, B, C)$;
6 Output $(\pi^{(b)}, \tau^{(1-b)})$.

In the real view of the verifier P_b, $\pi^{(b)}$ consists of (1) secret shares $A^{(b)}, x^{(b)}, y^{(b)}, z^{(b)}$, (2) random nonce $s^{(b)}$, (3) immediate results B, C, and (4) the distributed Fiat-Shamir randomness r. All these values are generated by the prover. (1), (2) and (3) are uniformly distributed. (4) is uniformly distributed due to the random nonce $s^{(b)}$ as described in [25].

In addition, $\tau^{(1-b)}$ consists of (1) secret shares $T_0^{(1-b)}$, $T_1^{(1-b)}$, $D^{(1-b)}$, (2) share of random oracle outputs $\hat{r}^{(1-b)}$ and r, and (3) immediate results B, C. (1) is uniformly distributed due to the secret share $z^{(b)}$. (2) is uniformly distributed due to random nonce $s^{(0)}, s^{(1)}$. Since $\hat{r} = r$ when the prover is honest and r is given to all verifiers, (2) reveals no new information. (3) is uniformly distributed due to the random x, y.

Therefore, the distribution output by Sim is identically to the view of verifier P_b, which concludes the proof of zero-knowledge. □

B.2 Proof of Theorem 2

Proof. We prove the Attr-PC construction in Fig. 3 satisfies the properties in Definition 2.

Correctness. 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 $[\![f_{\eta,(\beta,1)}]\!]$, we have $[\![\tilde{y}_i]\!] = (\beta, 1)$ if $j = \eta$ and $[\![\tilde{y}_i]\!] = (0, 0)$ otherwise. As such,

$$
\begin{aligned}
&C := C^{(0)} \cdot C^{(1)} := C_{\alpha,\rho} := e(g,g)^{ws} \\
&E := E^{(0)} \cdot E^{(1)} := E_{\alpha,\rho} := g^{s} \\
&B_d := B_d^{(0)} \cdot B_d^{(1)} := B_{\alpha,\rho,d} := g^{d\cdot q(d+1) - H'(a_d)\cdot s},\ \forall d \in \mathbb{Z}_m \\
&F_d := e(B_d, L) \cdot k_d := e(B_d, L) \cdot e(E, A_d), \forall d \in \mathbb{Z}_m
\end{aligned}
$$

where $s \in \mathbb{Z}_p$, q is a $(m-1)$-degree polynomial such that $q(0) = s$, $\{a_d\}_{d\in\mathbb{Z}_m}$ are attributes associated to the verification key $\mathsf{vk}_{\alpha,\rho} = (C_{\alpha,\rho}, E_{\alpha,\rho}, \{B_{\alpha,\rho,d}\}_{d\in\mathbb{Z}_m})$.

We show that if the secret key $\mathsf{sk} := (L, K, \{A_d\}_{d\in\mathbb{Z}_m})$ is generated by the same attributes $\{a_d\}_{d\in\mathbb{Z}_m}$ as $\mathsf{vk}_{\alpha,\rho}$ (i.e., $R(\mathsf{vk}_{\alpha,\rho}, \mathsf{sk}) = 1$), then the algorithm Verify outputs 1. As shown in Fig. 3, Verify outputs 1 if and only if (1) $\forall i \in \mathbb{Z}_m$, $\mathsf{ZK\text{-}PoK.Verify}(\tilde{\tau}^{(0)}, \tilde{\tau}^{(1)}) = 1$ and (2) $\forall i \in [0,1]$, $\mathsf{VDPF.Verify}(\tau_i^{(0)}, \tau_i^{(1)}) = 1$ and (3) $\forall i \in [2,4]$, $\tau_i^{(0)} = \tau_i^{(1)}$. The equality of (1) and (2) follows from the correctness of $\mathsf{ZK\text{-}PoK}$ and VDPF. For (3), the equality of $\tau_i^{(0)} = \tau_i^{(1)}, \forall i \in [2,3]$ follows from the special output $\tilde{y}_{\alpha,1} = 1$ and $\eta := \alpha\cdot\ell + \rho \in [\alpha\ell, \alpha\ell + \ell)$. To see why $\tau_4^{(0)} = \tau_4^{(1)}$, observe that

$$
\begin{aligned}
F_z &= \prod_{d\in\mathbb{Z}_m} F_d^{\Delta_{d+1,m}(0)} \\
&= \prod_{d\in\mathbb{Z}_m} (e(B_d, L) \cdot e(E, A_d))^{\Delta_{d+1,m}(0)} \\
&= \prod_{d\in\mathbb{Z}_m} e(g,g)^{tr\cdot q(d+1)\cdot\Delta_{d+1,m}(0)} \\
&= e(g,g)^{trs} \quad \text{by Lagrange interpolation formula}
\end{aligned}
$$

Therefore, we have $\dfrac{e(E,K)}{F_z \cdot C} = \dfrac{e(g,g)^{s(w+tr)}}{e(g,g)^{trs}\cdot e(g,g)^{ws}} = 1$ and $\dfrac{e(E^{(0)},K)}{F_z^{(0)}\cdot C^{(0)}} = \dfrac{F_z^{(1)}\cdot C^{(1)}}{e(E^{(1)},K)}$, i.e., $\tau_4^{(0)} = \tau_4^{(1)}$ as required.

Privacy. We construct an efficient simulator Sim for the view of any verifier P_b. We use the efficient simulator $\mathsf{Sim}_{\mathsf{VDPF}}$ and $\mathsf{Sim}_{\mathsf{ZK\text{-}PoK}}$ to generate the view of the VDPF output and the view of the $\mathsf{ZK\text{-}PoK}$ output, respectively. On input $(1^\lambda, b, \Lambda)$, 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. $\tau_2^{(1-b)} := \tau_2^{(b)}, \tau_3^{(1-b)} := \tau_3^{(b)}$
4. $L, K \leftarrow \mathbb{G}$, $k_i^{(b)} \leftarrow \mathbb{G}_t, \forall i \in \mathbb{Z}_m$
5. Compute $\tau_2^{(b)}, \tau_3^{(b)}, \tau_4^{(b)}$ using Λ, $f^{(b)}$, $\tilde{f}^{(b)}$, L, K and $\{k_i^{(b)}\}_{i\in\mathbb{Z}_m}$ according to Audit described in Fig. 3

6 $\tau_2^{(1-b)} := \tau_2^{(b)}, \tau_3^{(1-b)} := \tau_3^{(b)}, \tau_4^{(1-b)} := \tau_4^{(b)}$

7 $(\varphi_i^{(b)}, \tilde{\tau}_i^{(1-b)}) \leftarrow \mathsf{Sim}_{\mathsf{ZK\text{-}PoK}}(1^\lambda, b, \mathbb{G}, \mathbb{G}_t)$

8 $\pi^{(b)} := (\tilde{f}^{(b)}, L, K, \{k_i^{(b)}, \varphi_i^{(b)}\}_{i\in\mathbb{Z}_m})$

9 $\tau^{(1-b)} := (\{\tau_i^{(1-b)}\}_{i\in[0,4]}, \{\tilde{\tau}_i^{(1-b)}\}_{i\in\mathbb{Z}_m})$

10 Output $(\pi^{(b)}, \tau^{(1-b)})$

In the real view of the verifier P_b, $\pi^{(b)}$ consists of (1) the function share $f^{(b)}_{\alpha,(\beta,1)}$, (2) the values L, K, (3) the secret shares $k_i^{(b)}$, and (4) the corresponding ZK-PoK proofs $\varphi_i^{(b)}$ for $i \in \mathbb{Z}_m$. The distribution of (1) is computationally indistinguishable from the output of $\mathsf{Sim}_{\mathsf{VDPF}}$ due to the privacy of VDPF definition. (2) and (3) are uniformly distributed due to random r, w, t. The distribution of (4) is statistically indistinguishable from the output of $\mathsf{Sim}_{\mathsf{ZK\text{-}PoK}}$ due to the privacy of ZK-PoK definition. In addition, $\tau^{(1-b)}$ consists of (1) the VDPF tokens $\tau_2^{(1-b)}, \tau_3^{(1-b)}$, (2) the audit tokens $\tau_2^{(1-b)}, \tau_3^{(1-b)}$, and (3) the ZK-PoK tokens $\{\tilde{\tau}_i^{(1-b)}\}_{i\in\mathbb{Z}_m}$. The distribution of (1) is computationally indistinguishable from the output of $\mathsf{Sim}_{\mathsf{VDPF}}$. For (2), $\tau_2^{(1-b)}, \tau_3^{(1-b)}, \tau_4^{(1-b)}$ are equal to $\tau_2^{(b)}, \tau_3^{(b)}, \tau_4^{(b)}$. The distribution of (3) is statistically indistinguishable from the output of $\mathsf{Sim}_{\mathsf{ZK\text{-}PoK}}$.

We conclude that the distribution output by Sim is computationally indistinguishable from the distribution of the real view of P_b, as required.

Soundness. Assume that there exists an efficient adversary $\mathcal{A}$ and a non-negligible function δ such that:

$$\Pr[\mathsf{Soundness}_{\mathcal{A},b}(\lambda) = 1] \geq \delta(\lambda)$$

Extract $f_{\alpha,\beta}, \tilde{f}$ from the output by $\mathcal{A}$ in $\mathsf{Soundness}_{\mathcal{A},b}(\lambda)$. By the soundness property of VDPF, we can assume $f_{\alpha,\beta}$ and $\tilde{f}$ are point functions. Since H is a collision-resistant hash function, the equality check of $\tau_2^{(0)} = \tau_2^{(1)}$ and $\tau_3^{(0)} = \tau_3^{(1)}$ restricts $\mathcal{A}$ to outputting $\tilde{f}$ with the special input $\eta \in [\alpha\ell, \alpha\ell + \ell)$ and the special output $(\beta, 1)$, i.e., $\tilde{f} = f_{\eta,(\beta,1)}$. (See details in Appendix B.3 below.) It implies that verifiers obtain secret shares of $\mathsf{vk}_{\alpha,\rho}$ from Λ and $\rho := \eta - \alpha\ell$. In addition, for any $f \in \mathcal{F}$, there exists i such that $f \in \mathcal{Q}_i$ in Fig. 3, which implies that $\mathcal{A}$ wins only if $f_{\alpha,\beta} \notin T$.

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

1 $(\Lambda, \xi) \leftarrow \mathsf{PolicyGen}(1^\lambda, \mathcal{F})$

2 $\alpha' \leftarrow \mathbb{Z}_N, \rho' \leftarrow \mathbb{Z}_\ell, u_0, \ldots, u_{m-1}, s \leftarrow \mathbb{Z}_p$

3 Reset $\mathsf{vk}_{\alpha',\rho'} := (e(g^a, g^b), g^s, \{g^{u_d}\}_{d\in\mathbb{Z}_m})$ in Λ

4 Run $\mathcal{A}^{\mathsf{GetKey}}(1^\lambda, \Lambda)$ to get $f_{\alpha,\beta}, \pi := (f_{\eta,\beta}, K, L, -)$

5 **if** $\eta \neq \alpha'\ell + \rho'$, repeat step 6 (up to λ times)

6 $F_d^c := e(g^c, L^{u_d}) \cdot (g^c, A_d^s)$ for $d \in \mathbb{Z}_m$

7 Compute F_m^c using $\{F_d^c\}_{d\in\mathbb{Z}_m}$ according to Audit

8 $e(g,g)^{abcs} := e(g^c, K)^s / F_m^c$, by the requirement $e(g^s, K)/F_m = e(g^a, g^b)$ of valid proof

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

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 constraint list Λ, which is constructed by the output of $\mathsf{PolicyGen}(1^\lambda, \mathcal{F})$ and $\mathsf{vk}_{\alpha',\rho'} = (e(g^a, g^b), g^s, \{g^{u_d}\}_{d\in\mathbb{Z}_m})$. The view of $\mathcal{A}$ when run as a subroutine by $\mathcal{B}$ is distributed computationally indistinguishable to $\mathcal{A}$'s view in game $\mathsf{Soundness}_{\Lambda,\mathcal{A}}(\lambda)$, because: (1) $s_{\alpha',\beta'}$ is distributed identically to s; (2) the tuple $(e(g,g)^w, e(g,g)^s, e(g,g)^{ws})$ is computationally indistinguishable to the tuple $(e(g,g)^w, e(g,g)^s, e(g,g)^{ab})$ where a, b is uniformly random in $\mathbb{Z}_p$ (otherwise, the decisional BDH assumption is simply broken); (3) $\{B_{\alpha',\rho',d}\}_{d\in\mathbb{Z}_m}$ is distributed uniformly due to the random $(m-1)$-degree polynomial q. Since $\mathcal{B}$ wins if the output of $\mathcal{A}$ is valid $f_{\alpha',\beta}, \pi$ and $\eta \neq \alpha'\ell + \rho'$ for $f_{\eta,\beta}$ in π, we have that

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

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

B.3 Proof of Theorem 3

Proof. We prove the Tpl-PC construction in Fig. 4 satisfies the properties in Definition 2.

Correctness. 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 $\exists \rho, \beta \wedge \mathsf{rs}_{\alpha,\rho} = 0$ (i.e., $f_{\alpha,\beta} \in \mathcal{Q}_{\alpha,\rho}$), the algorithm Verify outputs 1. In Fig. 4, 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 correctness 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 $(1^\lambda, b, \Lambda)$, 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 Audit described in Fig. 4
4. $\tau_2^{(1-b)} := \tau_2^{(b)}, \tau_3^{(1-b)} := \tau_3^{(b)}$
5. Output $(\pi^{(b)} := \tilde{f}^{(b)}, \tau^{(1-b)} := \{\tau_i^{(1-b)}\}_{i\in[0,3]})$

The distribution output by Sim is computationally indistinguishable from the distribution of the real view of P_b, because: (1) the 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, when the prover is honest.

Soundness. Suppose for contradiction that there exists a PPT adversary $\mathcal{A}$ and a non-negligible function δ such that $\Pr[\mathsf{Soundness}_{\mathcal{A},b}(\lambda) = 1] \geq \delta(\lambda)$ Extract $f, \tilde{f}$ from the output by $\mathcal{A}$ in $\mathsf{Soundness}_{\mathcal{A},b}(\lambda)$. As described in Fig. 4, it holds that $R(\cdot,\cdot) \equiv 1$ in Tpl-PC scheme, which implies $T := \bigcup_{i\in\mathbb{Z}_N} \mathcal{Q}_i$. Hence, to win the game $\mathsf{Soundness}$, the adversary $\mathcal{A}$ need to make Verify output 1 and $\forall i, f \notin \mathcal{Q}_i$. In Fig. 4, 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 $\beta \neq \beta'$, the adversary $\mathcal{A}$ should find a string s such that there exist $a \in \mathbb{Z}_N$, $b \in \{0,1\}$, we have

$$H(s||a) = \bigoplus_{i\in\mathbb{Z}_N/\{a\}} H(y_i^{(b)} \oplus (\bigoplus_{j\in\mathbb{Z}_\ell} \tilde{y}_{i\cdot\ell+j}^{(b)})||i) \oplus \bigoplus_{i\in\mathbb{Z}_N} H(y_i^{(1-b)} \oplus (\bigoplus_{j\in\mathbb{Z}_\ell} \tilde{y}_{i\cdot\ell+j}^{(1-b)})||i),$$

$\{y_i^{(1-b)} \oplus (\bigoplus \tilde{y}_{i\cdot\ell+j}^{(1-b)})||i\} \cup \{s||a\} \neq \{y_i^{(b)} \oplus (\bigoplus \tilde{y}_{i\cdot\ell+j}^{(b)})||i\}$. It contradicts the XOR-collision-resistant 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., $\beta \wedge \mathsf{rs}_{\alpha,\rho} \neq 0$, $\mathcal{A}$ should find a string s' such that for an adversarially chosen $b \in \{0,1\}$, $H(s') := H(\bigoplus_{i\in\mathbb{Z}_N, j\in\mathbb{Z}_\ell} \mathsf{rs}_{i,j} \wedge \tilde{y}_{i\cdot\ell+j}^{(b)})$ but $s' \neq \bigoplus_{i\in\mathbb{Z}_N, j\in\mathbb{Z}_\ell} \mathsf{rs}_{i,j} \wedge \tilde{y}_{i\cdot\ell+j}^{(b)}$. It contradicts the collision-resistant of H. □

B.4 Proof of Theorem 4

Proof. We prove the IVDPF construction in Fig. 7 satisfies the properties in Definition 4.

Correctness. Except for the accumulated layer outputs $\{z_{i,0}, z_{i,1}\}_{i\in[n]}$, our IVDPF is same as the incremental DPF construction of Boyle *et al.* [14]. 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 $z_{i,\bar{\alpha}_i}^{(0)} + z_{i,\bar{\alpha}_i}^{(1)} = 0$ and $z_{i,\alpha_i}^{(0)} + z_{i,\alpha_i}^{(1)} = (\alpha_{[1,i]} \in X_{[1,i]}) \cdot \beta_i'$.

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

1 $s_0^{(b)} \leftarrow \{0,1\}^\lambda$, $\mathsf{ocw} \leftarrow \mathbb{G}$
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 $f^* := (s_0^{(b)}, (\mathsf{cw}_i, \mathsf{cs}_i, \mathsf{lcw}_i)_{i\in[n]}, \mathsf{ocw})$
5 $(\{y_x^*\}, \{z_{i,0}^*, z_{i,1}^*\}, \tau^*) \leftarrow \mathsf{IVDPF}(b, f^*, X)$

6 Output (f^*, τ^*)

In the real view of P_b, the IVDPF key $f^{(b)}$ consists of (1) random seed $s_0^{(b)}$ 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 Conv. Since P_b does not known the seeds of G and Conv, (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 $(\{y_x^{(b)}\}_{x\in X}, \{z_{i,0}, z_{i,1}\}_{i\in[n]}, \tau^{(b)}) \leftarrow \mathsf{Eval}(b, f_{\alpha,\beta}^{(b)}, X)$. Denote $C_i := \{0,1\}/\{1-j | z_{j,0}^{(0)} + z_{j,0}^{(1)} \neq 0, j \in \{0,1\}\}$. In addition, at every level i, each evaluator has a set of seeds and control bits, denoted as $\{\tilde{s}_{i,v}^{(0)}, \tilde{t}_{i,v}^{(0)},\}_{v\in X_{[1,i]}}$ and $\{\tilde{s}_{i,v}^{(1)}, t_{i,v}^{(1)}\}_{v\in X_{[1,i]}}$. The evaluators have the same correction seed cs_i. Let $\tilde{\tau}_{i,v}^{(b)} := H(\tilde{s}_{i,v}^{(b)} || t_{i,v}^{(b)} || v)$ and $\tau_{i,v}^{(b)} := \tilde{\tau}_{i,v} \oplus t_{i,v}^{(b)} \cdot \mathsf{cs}_i$ for $b \in \{0,1\}$. We consider two cases as follows:
Case 1: We prove that no PPT adversary $\mathcal{A}$ can construct VIDPF keys such that for $\left|\{x \in X | y_x^{(0)} + y_x^{(1)} \neq 0\}\right| > 1$ or $\exists i \in [n], |C_i| < 1$, it holds that $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1$. 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 $\tilde{s}_{i,u}^{(0)} || \tilde{t}_{i,u}^{(0)} \neq \tilde{s}_{i,u}^{(1)} || \tilde{t}_{i,u}^{(1)}$ and $\tilde{s}_{i,v}^{(0)} || \tilde{t}_{i,v}^{(0)} \neq \tilde{s}_{i,v}^{(1)} || \tilde{t}_{i,v}^{(1)}$, and $\forall x \in X_{[1,i]}$ we have $\tau_{i,x}^{(0)} = \tau_{i,x}^{(1)}$ to make $\mathsf{Verify}(\tau^{(0)}, \tau^{(1)}) = 1$ (due to the collision-resistant of H'). Because of the collision-resistant of H, we have $\tilde{\tau}_{i,u}^{(0)} \neq \tilde{\tau}_{i,u}^{(1)}$ and $\tilde{\tau}_{i,v}^{(0)} \neq \tilde{\tau}_{i,v}^{(1)}$. Hence, the adversary $\mathcal{A}$ needs to find cs_i such that $\mathsf{cs}_i = \tilde{\tau}_{i,u}^{(0)} \oplus \tilde{\tau}_{i,u}^{(1)} = \tilde{\tau}_{i,v}^{(0)} \oplus \tilde{\tau}_{i,v}^{(1)}$. It contradicts the XOR-collision-resistant of H.
Case 2: We prove that no PPT adversary $\mathcal{A}$ can construct VIDPF keys such that $\exists x \in X, (\exists i \in [n], x_i \notin C_i) \wedge (y_x^{(0)} + y_x^{(1)} \neq 0)$, 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 $\tilde{s}_{i,u}^{(0)} || \tilde{t}_{i,u}^{(0)} = \tilde{s}_{i,u}^{(1)} || \tilde{t}_{i,u}^{(1)}$, and $\mathsf{Conv}_{\{0,1\}^\lambda \times \mathbb{G}_i'}(\tilde{s}_{i,u}^{(0)}) \neq \mathsf{Conv}_{\{0,1\}^\lambda \times \mathbb{G}_i'}(\tilde{s}_{i,u}^{(1)})$. It contradicts the property of the PRG Conv.

In conclusion, the construction in Fig. 7 satisfies the soundness property of Definition 4. □

B.5 Proof of Theorem 5

Proof. We prove the IVDPF-PC construction in Fig. 8 satisfies the properties in Definition 2.
Correctness. 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 if the secret key $(\mathsf{sk}_0, \mathsf{sk}_1) \leftarrow \mathsf{SKGen}(\alpha, \xi)$, Verify outputs 1. In Fig. 8, Verify outputs 1 if and only if $\mathsf{IVDPF.Verify}(\tau_0^{(0)}, \tau_0^{(1)}) = 1$ and $\forall i \in [1,2], \tau_i^{(0)} = \tau_i^{(1)}$. The equality of the former follows from the correctness 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

$e(u, g^v) = e(\mathsf{sk}_0, g^{\mathsf{sk}_1}) = \sum_{i\in[n]} r_{i,\alpha_i} = \mathsf{vk}$. So $\frac{e(u, g^{v^{(0)}})}{\mathsf{vk}^{(0)}} = \frac{\mathsf{vk}^{(1)}}{e(u, g^{v^{(0)}})}$, i.e., $\tau_2^{(0)} = \tau_2^{(1)}$, as required.

Privacy. We construct an efficient simulator Sim for the view of any verifier P_b. We use the PPT simulator $\mathsf{Sim}_{\mathsf{IVDPF}}$ to generate the view of the IVDPF output. On input $(1^\lambda, b, \Lambda)$, 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 Audit
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 $(\pi^{(b)}, \tau^{(1-b)})$

The distribution output by 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) 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, when the prover is honest.

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

$$\Pr[\mathsf{Soundness}_{\mathcal{A},b}(\lambda) = 1] \geq \delta(\lambda)$$

Extract $f_{\alpha,\beta}$ from the output by $\mathcal{A}$ in $\mathsf{Soundness}_{\Lambda,\mathcal{A}}(\lambda)$. 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 verifiers 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 $(\Lambda, \xi) \leftarrow \mathsf{PolicyGen}(1^\lambda, \mathcal{F})$, $\alpha' \leftarrow \mathbb{Z}_N$
2 Reset $\mathsf{vk}_{i,\alpha_i'}$ in Λ, s.t., $\prod_{i\in[n]} \mathsf{vk}_{i,\alpha_i'} = e(g^a, g^b)$
3 Run $\mathcal{A}^{\mathsf{GetKey}}(1^\lambda, \Lambda)$ to get $f_{\alpha,\beta}, \pi := (u, v)$
4 **if** $\alpha' \neq \alpha$, repeat step 4 (up to λ times)
5 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
6 Output $r = e(g,g)^{abc}$

Similar to the proof of Theorem 2 (cf. Appendix B.2), $\mathcal{B}$ 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. □