Programmable Distributed Point Functions[*]

Elette Boyle[†] Niv Gilboa[‡] Yuval Ishai[§] Victor I. Kolobov[¶]

April 16, 2023

Abstract

A *distributed point function* (DPF) is a cryptographic primitive that enables compressed additive sharing of a secret unit vector across two or more parties. Despite growing ubiquity within applications and notable research efforts, the best 2-party DPF construction to date remains the tree-based construction from (Boyle et al, CCS'16), with no significantly new approaches since.

We present a new framework for 2-party DPF construction, which applies in the setting of feasible (polynomial-size) domains. This captures in particular all DPF applications in which the keys are expanded to the full domain. Our approach is motivated by a strengthened notion we put forth, of *programmable* DPF (PDPF), in which a short, input-independent "offline" key can be reused for sharing many point functions. We obtain the following results on PDPFs and their applications.

- **PDPF from OWF.** We construct a PDPF for feasible domains from the minimal assumption that one-way functions exist, where the second "online" key size is polylogarithmic in the domain size N.

Our approach offers multiple new efficiency features and applications:

- **Privately puncturable PRFs.** Our PDPF gives the first OWF-based *privately* puncturable PRFs (for feasible domains) with sublinear keys.
- $O(1)$**-round distributed DPF Gen.** We obtain a (standard) DPF with polylog-size keys that admits an analog of Doerner-shelat (CCS'17) distributed key generation, requiring only $O(1)$ rounds (versus $\log N$).
- **PCG with 1 short key.** Compressing useful correlations for secure computation, where one key is of minimal size. This provides up to exponential communication savings in some application scenarios.

1 Introduction

A *distributed point function* (DPF) [30, 14] is a cryptographic primitive that enables compressed sharing of a secret unit vector across two or more parties. More concretely, a two-party DPF allows one to split any *point function* f_α (i.e., for which $f_\alpha(x) = 1$ if $x = \alpha$, and 0 otherwise[1]) into succinctly described functions f_0, f_1, that individually hide f_α, and which support a simple additive per-input reconstruction $f_\alpha(x) = f_0(x) + f_1(x)$.

DPFs with function share f_i size (sometimes referred to as "key size") comparable to the truth table of f_α are trivially achievable, by simply taking the function shares f_i to be an additive secret sharing of the full truth table itself. Efficient constructions with small key size, roughly logarithmic in the domain size of f_α, have been built from one-way functions [30, 14].

[*]This is a full version of [16].

[†]IDC Herzliya, Israel and NTT Research, USA, elette.boyle@idc.ac.il.

[‡]Ben-Gurion University, Israel, gilboan@bgu.ac.il.

[§]Technion, Israel, yuvali@cs.technion.ac.il.

[¶]Technion, Israel, tkolobov@cs.technion.ac.il.

[1]Slightly more generally, $f_{\alpha,\beta}$ with $f_{\alpha,\beta}(\alpha) = \beta$ for $\beta \in \{0, 1\}$.

The appealing compressing structure of DPF constructions has enabled a wide range of cryptographic applications, ranging from Private Information Retrieval (PIR) [21, 30], to anonymous messaging systems [23], secure computation for RAM programs [28] and programs with mixed-mode operations [15, 8], and recently Pseudorandom Correlation Generators [9, 15, 13] for expanding small correlated seeds into large pseudorandom instances of cryptographic correlations, with applications to secure computation and beyond.

In many (if not most) of these applications, the parties perform a full evaluation of the DPF function shares, on every input within the domain of the function f_α. This means that, in particular, the necessary DPF constructions are only relevant for relatively small, polynomial-size domains.

The growing list of applications has provided significant motivation for deeper study of the DPF primitive, including alternative constructions and careful fine-tuning of efficiency. However, despite notable research efforts, the best 2-party DPF construction to date (even concrete constants) remains the tree-based construction from [14]. In addition, no significantly new approaches toward construction have emerged since this time.

1.1 Our Results

We present a new approach for DPF construction, whose structure dramatically differs from existing DPFs, and which offers new efficiency features and applications in the setting of feasible (polynomial)-size domains.

1.1.1 Programmable DPF.

Perhaps the primary downside of DPFs is that their security guarantees inherently require the existence of *two or more* non-colluding parties who receive shares f_0 and f_1 of the secret function. For example, DPFs yield solutions to the problem of *two*-server PIR, but seem useless for single-server PIR. Unfortunately, this non-collusion trust assumption is to some degree unavoidable for efficient solutions. For problems like PIR, for example, it is known that *single*-server cheap (symmetric-key) solutions simply cannot exist [27]. However, the two-server state of affairs has a further downside beyond the assumption of trust. Given two servers operating, DPF-based solutions incur twice as much computation, communication, and coordination costs between parties than if a single server could suffice.

Given the barrier of efficient single-server solutions, we consider a next best alternative: a form of *"1.5-server"* DPF, or what we will refer to as a *programmable* DPF. The idea is that participation and cost to one of the two servers will be pushed to minimum, thus incurring the burden of only "half" a server. Concretely, in a programmable DPF scheme, the share f_0 given to the first server is simply a random (i.e., "programmed") 128-bit string,[2] independent of the choice of—or in some cases, even the *parameters* of—the secret point function being shared. For example, one can execute the role of the first server across several applications via a public service.

Naive PDPF. As a baseline, consider a naive construction of PDPF: The offline key is a standard PRF key, and the online key is simply a domain-size string which together with the full-doman expansion of the PRF form additive secret shares of the desired truth table. The runtime of key generation is linear in the domain size N (which, looking ahead, will match that of our construction). However, the online key size is also linear in N, which we will succeed to compress exponentially. In addition, for the case of sharing a random point function, we will also obtain exponential improvement in the online key generation.

Related notions. Our PDPF goal is related to two existing notions from the literature: privately puncturable PRFs [6], and two-server PIR in an offline/online model recently studied by Corrigan-Gibbs and Kogan [24].

Privately puncturable PRFs are pseudorandom functions (PRFs) that support generation of punctured keys which enable evaluation of the PRF on all but a single punctured input x^*, and which further *hide* the identity of x^*. (Single-key) privately puncturable PRFs are in fact implied by programmable DPFs, by taking

[2]Or rather, λ bits, where λ is the security parameter.

the master PRF key to be the first-server DPF share, and generating a punctured key at x^* by computing a second-server DPF share for the function $f_{\alpha,\beta}$ with $\alpha = x^*$ and $\beta \leftarrow \{0,1\}$ selected at random. In turn, privately punctured PRF constructions can provide a direction toward programmable DPFs. However, the only existing instantiations of privately puncturable PRFs make use of heavy public-key cryptography machinery, and provide heavy costs for concrete applications [5, 20, 19, 42]. There is also no clear way "scale down" these constructions to a polynomial-size domain in a way that circumvents these issues.

Analogous to the "half server" of programmable DPF, the offline/online 2-server PIR protocols of [24, 44] consider a setting where first server's query and response (analogous to our first-server DPF key) can be computed *offline*, before the target input (analogous to the punctured point x^*) is specified. However, the resulting schemes do not yield the stronger target of a DPF. Indeed, the closest object they construct supports a nonlinear reconstruction procedure more complex than simple addition, which precludes a large subset of DPF applications requiring this structure (such as secure aggregation). In addition, [44] uses public-key cryptography.

Given the collective state of the art, no solutions exist for nontrivial programmable DPF without public-key cryptography, even for the restricted case of polynomial-size domains.

1.1.2 Programmable DPF on small domains from OWF.

We present a 2-party programmable DPF (PDPF) construction for polynomial-size domains, relying on the minimal assumption that one-way functions exist.

We begin with a basic construction which has a non-negligible privacy error ϵ, which appears as a factor of $\log(1/\epsilon^2)$ in the key size and $1/\epsilon^2$ in full-domain evaluation, and which provides appealing concrete efficiency. For this reason, we express the result statement in terms of a length-doubling pseudorandom generator (whose existence is equivalent to one-way functions). We remark that small constant privacy error is motivated in many applications in the context of concrete efficiency, such as those anyway offering differential privacy guarantees (e.g., use of DPFs for private aggregate statistics in [4]).

For the final feasibility result, we then reduce this to negligible error via a nontrivial amplification procedure. Combining these two theorems provides a construction of PDPF with polylogarithmic online key size, from one-way functions.

Theorem 1 (1/poly-secure PDPF on small domains - Informal)**.** *Given length doubling PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$, there exists a computationally ϵ-secure Programmable DPF for point functions $f_\alpha : [N] \to \{0,1\}$ over output group $\mathbb{G} = \mathbb{Z}$, with online key size $|k_1| = \lambda \log(N/\epsilon^2)$. Moreover:*

- *Key generation makes $(2N \log(N/\epsilon^2))/\lambda$ invocations of G, and*
- *Full domain evaluation makes $(2N \log N)/(\epsilon^2 \lambda)$ invocations of G.*

Theorem 2 (Security amplification - Informal)**.** *Suppose there exists a small-domain computationally $1/P(\lambda)$-secure PDPF for any polynomial P. Then there exists a small-domain PDPF with negligible security error.*

Corollary 1 (PDPF from OWF - Informal)**.** *Assuming the existence of OWF, there exists a PDPF for point functions $f_\alpha : [N] \to \{0,1\}$ with online key size $\text{poly}(\lambda, \log N)$, where the runtime of key generation and full domain evaluation is nearly linear in N.*

We turn to discuss the advantages and limitations of our PDPF constructions.

Small domains: Applications and non-applications. Note that the since the running time of the PDPF key generation and evaluation algorithms is comparable to the domain size N, we are restricted to polynomial-sized domains. As an additional point of interest, our techniques do not admit a more efficient single-point evaluation algorithm than a full-domain evaluation. We leave open the existence of a OWF-based PDPF construction where the running time of key generation and single-point evaluation is only $\text{poly} \log N$.

On the bright side, many DPF applications only require a small (polynomial-size) domain. This captures a motivated range of applications and implemented systems, including:

1. *Private "reading"* applications, such as PIR, or private tag-based search for tag space of modest size. For example, the Popcorn system [32] ran 2-server PIR on $N = 8,000$ Netflix movies.

2. *Private "writing"* applications, such as secure distributed storage [41], voting, and aggregation. This includes Prio-style [22] applications for private collection of aggregate statistics, and Riposte [23], Blinder [1] and Spectrum [40] for anonymous messaging.

3. *Pseudorandom correlation generators (PCGs) for useful correlations.* Relevant correlation examples include "silent" generation of permuted one-time truth table correlations, oblivious linear evaluation (OLE), or authenticated multiplication triples [13, 11] (for some simpler correlations the full power of DPF is not needed – see below).

4. *Mixed-mode secure computation with small-domain gates.* DPFs and their derivatives, most notably *distributed comparison functions* (DCF) (i.e. secret sharing functions of the form $f_{\alpha}^{\leq}$ that evaluate to 1 on all inputs $x \leq \alpha$), yield a method for highly efficient secure computation of certain types of non-arithmetic gates in the preprocessing model [15]. A DCF can be implemented by a logarithmic number of DPF invocations, one for each prefix of the shared point. However, in our small-domain construction the communication and computation for this DCF implementation essentially match those of a single DPF since both require full domain evaluation. Small-domain DPFs and DCFs suffice, e.g., for secure evaluation of zero-test, comparison/threshold, ReLU, splines, or finite-precision fixed-point arithmetic gates, on moderate-size inputs [15, 8]. We remark that small domain sizes often arise naturally in settings such as privacy-preserving machine learning, where computations are frequently run in low precision.

Aside from the last (Item 4), each of the above application frameworks further requires the parties to perform a full-domain evaluation of the corresponding DPF function shares, *inherently* limiting the desired DPF tools to small domains.

The programmable feature of our PDPF, where the offline key is short and reusable, offers beneficial properties in the above settings. For example, for pseudorandom correlation generation, this enables a central server to have a *single short* PCG key for generating authenticated multiplication triples or truth-table correlations with many different users, requiring total storage of only 128 bits improving over present solutions that require the server to store approximately 1MB *per user*. Such a "short-key PCG" can make a big difference in certain applications of secure two-party computation. For instance, this is the case when during a setup phase one of the two parties can be temporarily trusted. In this case, she can generate a pair of PCG seeds, send the short (128-bit) seed to the other party, and keep the longer one to herself. We discuss this application in more detail in Section 1.1.3.

There are, of course, application settings in which small-domain DPFs are not relevant. Prominent examples include:

1. *Private keyword search*, corresponding to PIR-type private queries where the space of possible inputs (e.g., universe of keywords) is large.

2. *Simpler pseudorandom correlation generators*, such as "silent" oblivious transfer, vector OLE, or (unauthenticated) multiplication triples, do not require the full 2-sided guarantees (so-called "puncturable PRFs" suffice).

3. *Mixed-mode secure computation with large-domain gates.* The above mixed-mode application is viable also for large domains, in which case our small-domain DPFs do not provide a solution. This includes instances of the above gates over large inputs.

Concrete efficiency. In Table 1 we compare the efficiency of our programmable DPF construction to a "naive" construction with $O(N)$ key size, for domain size N (see Section 5 for more details). The comparison is done with output group $\mathbb{Z}$ and with payloads in $\{0,1\}$, capturing a typical aggregation scenario. We compare these solutions with respect to key size, and estimate the running time of an AES-based implementation by using a standard benchmark of $1.8 \cdot 10^8$ length-doubling PRG calls per second on a single core.

To give one data point, for a domain of size $N = 100,000$ and security error $\epsilon = 2^{-8}$, the naive construction has $97.7KB$ key size, and the running time for either key generation or full domain evaluation is $72.1\mu s$, while our construction achieves $0.5KB$ key size, $548.3\mu s$ running time for key generation, and $1.6 sec$ running time for full domain evaluation[3]. In another data point, where $N = 20,000$ and $\epsilon = 2^{-6}$, the naive construction yields $14.7\ KB$ key size and running time of $12.4\ \mu s$ for both key generation and full domain evaluation, while our constructions has $0.4\ KB$ key size, $68.7\ \mu s$ running time for key generation, and $17.3\ ms$ running time for full domain evaluation. Note that in applications that only require a random point α, the cost of Gen can be substantially smaller: $0.006\,\mu s$ for a domain of size $N = 100,000$ and security error $\epsilon = 2^{-8}$, and $0.005\,\mu s$ for $N = 20,000$ and $\epsilon = 2^{-6}$.

To conclude, for small input domains and small (but non-negligible) privacy levels ϵ, our construction offers a big advantage in key size, a moderate slowdown on the client side (running the key generation), and a more significant slowdown on the server side (running the full domain evaluation). Overall, we expect it to be attractive for applications where the client's communication is the most expensive resource.

Comparison to standard DPF. Compared to a standard two-party DPF, our PDPF construction offers several qualitative advantages which can be appealing in the following settings:

- When simplifying the interaction pattern is important. For some DPF applications, the "1.5-server" feature means that online interaction only involves a single message from the client to the online server (and no interaction between servers). This offers several advantages for practical systems such as avoiding the dependence on two online, synchronised servers, reducing network latency, and also hiding the identity of the offline server, rendering the non-collusion assumption more realistic.
- When the client can play the role of the online server, as in the "trusted-offline PCG" application discussed in Section 4.3. In such cases, a PDPF yields a near-exponential improvement in the *total* communication cost, since it only requires a one-time communication of an offline key which is reused many times without further interaction.
- When distributed key generation is carried out over a high-latency network, the constant-round black-box protocol from Section 4.2 can offer significant speedup.

All of the above advantages seem relevant for practical use cases. Our PDPF construction has a reasonable concrete overhead (to be discussed below) when settling for small but non-negligible values of ϵ, comparable to the acceptable practices for differential privacy.

Other than the application scenarios described above, our current PDPF construction is less practical than existing DPF constructions. First, it cannot offer negligible privacy error ϵ with good concrete efficiency; second, the running time of Gen and (single-point) Eval scale linearly (rather than logarithmically) with the domain size; finally, it has worse dependence (multiplicative rather than additive) on the size of the payload β. These gaps are smaller in applications that require a full-domain evaluation $\mathsf{EvalAll}$, or alternatively only require key generation for a random point α (see below).

Comparing the key size of the two constructions, note that the size of the keys in PDPF is $\log(N/\epsilon^2)$ PRG seeds for the online party and just a single PRG seed for the offline party, while the key size of both parties in standard DPF is roughly $\log(N)$ PRG seeds. Ignoring the qualitative advantages of PDPF over DPF, the total client communication, or total key size, of PDPF is smaller by almost a factor of two for concretely relevant parameters.

In the case of a *random-input* PDPF, the client computation becomes roughly equal to that of a standard DPF, i.e. dominated by $\log(N)$ calls to a PRG, since the client generates one key which is a seed of a GGM

[3]In fact, the naive construction, as mentioned in Section 1.1.1, can provide a *negligible* privacy error for small output groups. Nevertheless, in aggregation-type applications, over output group $\mathbb{Z}$, we get a constant privacy error. See Remark 4 for more details.

PRF and another key which is the same PRF punctured at a random point. A random-input PDPF is good enough for some applications, such as distributed key generation, on which we elaborate in Section 4.2. There, a random-input PDPF and can be converted to a chosen-input DPF by sending a $\log(N)$-bit offset to the offline server.

While our PDPF construction has higher overhead as the output size grows compared to a standard DPF, in Proposition 3 we provide an optimization to our construction for big payloads beyond the naive approach of executing a separate PDPF instance for every bit of the payload.

1.1.3 Applications.

We explore three applications of our programmable DPF construction and associated techniques: (1) Privately Puncturable PRFs (on polynomial size domains); (2) (Standard) Distributed Point Functions that admit particularly efficient secure distributed key generation protocols; and (3) A new application regime of *trusted-offline* pseudorandom correlation generators. We additionally explore an optimization toward DPFs with larger payloads.

Privately puncturable PRFs (on small domains). As discussed, our construction directly implies the first nontrivial *privately puncturable PRF* for domain size $N = \mathrm{poly}(\lambda)$ under the minimal one-way function assumption. Here, nontriviality corresponds to requiring the key size of a (privately) punctured key that is sublinear in the truth table output size.

Even given the restriction to feasible domain sizes, this constitutes the first such construction without relying on structured public-key assumptions such as the Learning with Errors assumption or multi-linear maps [5, 20, 19, 42].

Proposition 1 (Privately puncturable PRF - Informal)**.** *Assuming the existence of OWF, there exist (selectively secure, 1-key) privately puncturable PRF (P-PPRF), where the runtime of punctured key generation and evaluation is quasilinear in the domain size M, and with punctured key size* $\mathrm{poly}(\lambda, \log M)$*.*

DPF with constant-round black-box distributed key generation. In any application of (standard) DPFs where the role of "client" is jointly executed across parties—including secure computation for RAM programs [28] or mixed-mode operations [15, 8], use of pseudorandom correlation generators for secure computation preprocessing [9, 15, 13], and more—the Gen algorithm of the DPF must in turn be executed distributedly via a secure computation protocol. Minimizing the costs of this procedure is a highly desirable target.

This was highlighted by the work of Doerner and shelat [28], which identified that the low cost of distributed DPF Gen makes it a strong approach for secure computation of RAM programs. They presented a distributed DPF Gen protocol, which remains the most efficient to date, requiring computation time linear in the DPF domain size N, and runs in $\log N$ sequential communication rounds, but which crucially makes only *black-box* use of oblivious transfer and a pseudorandom generator. In contrast, alternative approaches each require the expensive secure evaluation of (many instances of) a circuit evaluating the PRG.

In particular, for any DPF with key size polylogarithmic in the domain size N,[4] no protocol exists for distributed Gen which is black-box in the underlying cryptographic tools and lower than $O(\log N)$ round complexity.

The techniques behind our PDPF give the first DPF (for feasible domains) which simultaneously achieves key size polylogarithmic in N, and admits a distributed Gen protocol that makes only black-box use of OT and a PRG, executing in *constant round complexity.* More concretely, we show that 5 rounds suffice.

Proposition 2 (Constant-round distributed Gen - Informal)**.** *There exists a small-domain DPF* $(\mathsf{Gen}, \mathsf{Eval})$*, with key size* $\mathrm{poly}(\lambda, \log N)$*, where* Gen *on secret-shared α, β can be implemented by a constant-round (5-round) protocol making only a black-box use of oblivious transfer and a pseudorandom generator.*

[4]DPFs with significantly worse key size N^ϵ for constant $\epsilon > 0$ can be built with lower depth Gen, e.g. by "flattening" the tree structure of current best DPF constructions.

As with our PDPF constructions, the runtime of our DPF Eval algorithm will be linear in the domain size N. Note, however, that the application of DPF within secure computation of RAM programs anyway requires $\mathsf{EvalAll}$ as opposed to individual Eval operations (where we achieve the same linear complexity). In addition, our resulting DPF Gen procedure will only be logarithmic in N. This will be result of modifying the PDPF, adding a short second "offset" message to be added to the key k_0 after the choice of the secret point function $\hat{f}_{\alpha,\beta}$. This extra step adds minor cost in regard to computation and key size, but means the resulting construction is a DPF and not a *programmable* DPF which in particular requires the first key k_0 to be independent of the point function to be shared.

Compressing DPF corelations. Standard DPFs have a variety of applications in the context of secure 2-party computation (2PC). For instance, they serve as crucial building blocks for concretely efficient 2PC of RAM programs [28] or for pseudorandom correlation generators (PCGs) of truth-table correlations [12] and (authenticated) multiplication triples [11]. Evaluating large circuits or multiple instances necessitates several DPF correlations. In particular, this strongly motivates the goal of generating many independent instances of a random DPF correlation with low communication cost. However, there are no known practical methods for achieving this.

We observe that PDPF inherently provides a solution for generating many such instances, where the size of one key scales with the number of instances, but *one key is short*.

In turn, our PDPF provides a solution to the above problem within a subset of interesting applications, captured by the following "trusted-offline" setting for 2PC. In an offline phase, Alice owns a long-term secret s (say, a secret key for encryption, identification, or signature). To eliminate a single point of failure, she splits s into two shares, s_A and s_B, sending s_B to Bob and keeping s_A to herself. She then erases all information except s_A. In the online phase, the parties receive online inputs P_i (resp., ciphertexts to decrypt, nonces for identification, or messages to sign) and wish to securely compute $f(s, P_i)$ for $i = 1, 2, \ldots, t$.

The key observation is that in the above setting, Alice can be fully trusted during the offline phase, since if she is corrupted at this phase (before erasing s) then the long-term secret is entirely compromised. In fact, if P_i is public, then s is the only secret in the system. For this reason, we can also trust Alice to generate pairs of DPF keys (k_0^j, k_1^j) in the offline phase, offload the keys k_0^j to Bob, and keep k_1^i to herself. However, when Alice wants to generate many DPF instances for the purpose of evaluating many g-gates, this has high communication cost.

A PDPF can provide a dramatic efficiency improvement in this scenario, where Alice needs only to send the single *short* PDPF key to Bob, and simply store the longer key locally. This reduces the communication requirements of existing solutions within this setting by an exponential factor.

Big payload optimization. Some applications of DPF explicitly require the point function payload to be larger than a single bit, e.g. an element in $\mathbb{Z}_{2^\ell}$, and to be random. A natural adaptation of our technique to this setting is to repeat the programmable DPF scheme with binary outputs ℓ times, once for each bit, and then locally map the outputs to elements in $\mathbb{Z}_{2^\ell}$. However, evaluation using this approach suffers from an $O(\ell^3)$ computational overhead compared to a binary programmable DPF achieving the same security[5].

We propose an optimization which maintains key size and reduces the computational overhead by $O(\ell)$ compared to the repetition method. In more detail, each PRF value is a pair of a point x in the input domain $[N]$ and a value $y \in \mathbb{Z}_{2^\ell}$. One key of the programmable DPF is again the short PRF key, while the second key is punctured at $O(\ell)$ points which evaluate to $(\alpha, y_i), i = 1, \ldots, O(\ell)$. The DPF evaluation at each point x is the sum of all y_i such that the PRF (or punctured PRF) evaluate to (x, y_i) at some point. This approach leads to the following:

Proposition 3 (Big payload optimization - Informal)**.** *Given length doubling PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$, there exists a computationally ϵ-secure PDPF for point functions $f_\alpha : [N] \to \mathbb{Z}_{2^\ell}$, with online key size*

[5]In this approach, to get statistical error of ϵ we need to reduce the value of ϵ in each of the ℓ instances by a factor of ℓ. Since the computational cost per instance depends quadratically on $1/\epsilon$, this results in a total slowdown (compared to the 1-bit baseline) of $\ell \cdot \ell^2 = \ell^3$.

$|k_1| = O\left(\lambda t \log \frac{tN}{\epsilon^2}\right)$ *for* $t = \ell + 2log\frac{1}{\epsilon}$. *The number of invocations of* G *in the key generation algorithm is* $O(tN \log \frac{t}{\epsilon^2})$*, and in the full domain evaluation algorithm it is* $O(\frac{Nt^2}{\epsilon^2})$.

1.2 Overview of Techniques

We now proceed to describe our techniques in greater detail. We focus here on the core construction of programmable DPF from OWF. We refer the reader to the main body for further detail on the related applications.

1/poly-Secure PDPF. We begin by describing our construction of a computationally secure PDPF, which takes inspiration from the *puncturable pseudorandom sets* of Corrigan-Gibbs and Kogan [24].

Our construction relies on an underlying tool of Punctuarable Pseudorandom Functions (PPRF) [7, 36, 17]. Puncturable PRFs are an earlier-dating, weaker variant of privately puncturable PRFs discussed above, which similarly have the ability of generating *punctured* keys k_p from a master PRF key k enabling evaluation on all but a punctured input x_p. Even given the punctured key k_p, the output of the PRF at input x_p remains pseudorandom. Unlike privately puncturable PRFs, no hiding requirement is made for the identity of the punctured input x_p given the punctured key k_p, which makes the goal significantly easier to achieve. Such primitives can be constructed in a simple manner based on one-way functions via a GGM [31] tree [7, 36, 17].

Our construction proceeds roughly as follows. Consider the first party in the programmable DPF. The first (programmable) key of the DPF is simply the master key k for a PPRF whose output space is the *input* space for the DPF, $[N]$. The PRF input space D will be selected in the discussion following, as a function of the desired privacy error. (In particular, larger domain will yield smaller error, but higher complexity.)

In order to expand its DPF key to a full-domain evaluation on the input domain $[N]$, the first party begins by evaluating its PRF tree on all inputs. Recall that each leaf of the PRF evaluation tree is now labeled by some element of $[N]$. For each $x \in [N]$, the corresponding DPF output evaluation $f_1(x)$ is defined to be the integer *number of occurrences* of the value x within the leaves of the PRF tree: i.e., the number of values ζ in the input space D of the PRF for which $PRF_k(\zeta) = x$.

Pictorially, each PRF leaf evaluation can be viewed as a "ball" thrown into one of N bins, labeled $1, \ldots, N$. Evaluating on the complete PRF tree (given the master key k) results in a histogram, of number of balls per bin, which constitutes the evaluated DPF output share values.

The second key in the programmable DPF is generated given the target point function f_{α^*} we wish to share. Observe that (for payload $\beta = 1$) the goal is to recreate the same "balls in bins" histogram as above, but with 1 less ball in the $\alpha^* \in [N]$ bucket.[6] Indeed, if this can be achieved, then the parties' shares differ by 0 in all places apart from α^*, and precisely by 1 at α^*. To do so, the second server will be given the PRF key *punctured* at a random input x_p whose PRF output is α^*. In effect, one (random) ball is removed from the α^* bin.

Correctness of the construction holds as above. But, we find ourselves encountering a serious security challenge. While clearly the first party's share is independent of the secret function f_{α^*}, security against the second party must somehow rely on hiding the punctured PRF evaluation given access to a punctured key. However, in a puncturable PRF, pseudorandomness is only guaranteed when the punctured input is chosen *independently* of the PRF evaluation values. In contrast, the input we puncture is selected based on the PRF evaluations. In fact, the issue is even worse. Even the stronger notion of adaptive security of PPRF does not suffice, where the punctured input can be selected as function of the PRF evaluations on *other* inputs. In our construction the punctured input is chosen as function of *its own* evaluation—in general, one cannot hope to achieve this kind of security.

Indeed, the resulting construction does not provide negligible leakage in privacy. This corresponds to the (non-negligible, efficiently identifiable) statistical difference in the N histogram counts when throwing a polynomial number of balls and then removing a ball from one bin. This statistical difference can be

[6]To account for the fact that the payload could be $\beta = 0$, we actually introduce dummy bucket $N + 1$ to the PRF output space; removing a ball from this bucket means that all $[N]$ buckets remain equal across parties.

decreased by increasing the total number of balls thrown: this corresponds directly to a larger choice of the puncturable PRF domain D. Roughly, increasing D by a factor of $c > 1$ cuts the error by a factor of $1/\sqrt{c}$.

We provide a tight analysis of privacy error via a careful sequence of hybrid experiments, where the α^*-output-punctured key is ultimately replaced by a key punctured at a random independent input. Each step within the proof introduces negligible error, aside from one: in which we move from a PRF key where we puncture an input with a random *output value* (i.e., the DPF construction for a random α^*, to one where we puncture a random *input*.

It is interesting to observe that the construction is sensitive to specific design choices. For example, slightly modifying the above procedure to instead puncture the *first* input whose output α^* (instead of a random such input) yields a serious attack: given the punctured PRF key, the second party can directly infer for all values $\alpha' \in [N]$ appearing as PRF evaluations *before* the punctured point that $f_{\alpha'}$ is *not* the secret shared point function.

Amplification. To amplify a DPF with 1/poly privacy error into one with negligible error, we apply a privacy amplification technique based on a locally random reduction. The idea is to lift the input domain to a codeword in a Reed-Muller code and decode along a random low-degree curve. This effectively reduces a single DPF with secret input α to a small number of instances of DPF with secret inputs α_i, where the α_i are $\operatorname{poly}\log\lambda$-wise independent. By combining a "statistical-to-perfect" lemma from [37, 29] with a computational hardcore lemma of [38], the 1/poly leakage on each α_i can be argued to be no worse than completely leaking each α_i with small probability, which by $\operatorname{poly}\log\lambda$-wise independence suffices to hide α except with negligible probability.

2 Preliminaries

Notation. For $N \in \mathbb{N}$ we let $[N] = \{1, \ldots, N\}$. We denote the inner product of two vectors u and v of the same length by $\langle u, v\rangle = \sum_i u_i v_i$. We denote by negl a negligible function.

Probability. For two distributions D_1, D_2 we denote by $d(D_1, D_2) = \frac{1}{2}\sum_\omega |\Pr_{D_1}[\omega] - \Pr_{D_2}[\omega]|$ their statistical distance. We denote by U_ℓ uniformly distributed random strings of length ℓ.

Groups. We represent an Abelian group $\mathbb{G}$ of the form $\mathbb{G} = \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_\ell}$, for prime powers $q_1, \ldots, q_\ell$ by $\hat{\mathbb{G}} = (q_1, \ldots, q_\ell)$ and represent a group element of $\mathbb{G}$ by a sequence of ℓ non-negative integers. Unlike previous DPF definitions, here we will also consider infinite groups, using $q_i = \infty$ for the group of integers $\mathbb{Z}$.

Point functions. Given a domain size N and Abelian group $\mathbb{G}$, a *point function* $f_{\alpha,\beta} : [N] \to \mathbb{G}$ for $\alpha \in [N]$ and $\beta \in \mathbb{G}$ evaluates to β on input α and to $0 \in \mathbb{G}$ on all other inputs. Unlike previous DPF definitions, here we will also consider the case where the output β is guaranteed to be taken from a subset $\mathbb{G}' \subseteq \mathbb{G}$, where the subset $\mathbb{G}'$ can be leaked. This extension is especially useful when $\mathbb{G} = \mathbb{Z}$, in which case we will typically let $\mathbb{G}' = \{0, 1\}$. When $\mathbb{G}'$ is omitted, we assume $\mathbb{G}' = \mathbb{G}$. We denote by $\hat{f}_{\alpha,\beta} = (N, \hat{\mathbb{G}}, \hat{\mathbb{G}}', \alpha, \beta)$ the representation of such a point function.

2.1 Distributed Point Functions

We begin by defining a slightly generalized notion of distributed point functions (DPFs), which accounts for the extra parameter $\mathbb{G}'$.

Definition 1 (DPF [30, 14])**.** *A (2-party)* distributed point function (DPF) *is a triple of algorithms* $\Pi = (\mathsf{Gen}, \mathsf{Eval}_0, \mathsf{Eval}_1)$ *with the following syntax:*

- $\mathsf{Gen}(1^\lambda, \hat{f}_{\alpha,\beta}) \to (k_0, k_1)$*: On input security parameter* $\lambda \in \mathbb{N}$ *and point function description* $\hat{f}_{\alpha,\beta} = (N, \hat{\mathbb{G}}, \hat{\mathbb{G}}', \alpha, \beta)$*, the (randomized) key generation algorithm* Gen *returns a pair of keys* $k_0, k_1 \in \{0,1\}^*$*. We assume that* N *and* $\mathbb{G}$ *are determined by each key.*

- $\mathsf{Eval}_i(k_i, x) \to y_i$*: On input key* $k_i \in \{0,1\}^*$ *and input* $x \in [N]$ *the (deterministic) evaluation algorithm of server* i*,* Eval_i *returns* $y_i \in \mathbb{G}$*.*

We require Π *to satisfy the following requirements:*

- ***Correctness:*** *For every* λ*,* $\hat{f} = \hat{f}_{\alpha,\beta} = (N, \hat{\mathbb{G}}, \hat{\mathbb{G}}', \alpha, \beta)$ *such that* $\beta \in \mathbb{G}'$*, and* $x \in [N]$*, if* $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, \hat{f})$*, then* $\Pr\left[\sum_{i=0}^{1} \mathsf{Eval}_i(k_i, x) = f_{\alpha,\beta}(x)\right] = 1$.

- ***Security:*** *Consider the following semantic security challenge experiment for corrupted server* $i \in \{0,1\}$*:*

  1. *The adversary produces two point function descriptions* $(\hat{f}^0 = (N, \hat{\mathbb{G}}, \hat{\mathbb{G}}', \alpha_0, \beta_0), \hat{f}^1 = (N, \hat{\mathbb{G}}, \hat{\mathbb{G}}', \alpha_1, \beta_1)) \leftarrow \mathcal{A}(1^\lambda)$*, where* $\alpha_i \in [N]$ *and* $\beta_i \in \mathbb{G}'$*.*
  2. *The challenger samples* $b \xleftarrow{\$} \{0,1\}$ *and* $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, \hat{f}^b)$*.*
  3. *The adversary outputs a guess* $b' \leftarrow \mathcal{A}(k_i)$*.*

  Denote by $\mathrm{Adv}(1^\lambda, \mathcal{A}, i) = \Pr[b = b'] - 1/2$ *the advantage of* $\mathcal{A}$ *in guessing* b *in the above experiment. For circuit size bound* $S = S(\lambda)$ *and advantage bound* $\epsilon(\lambda)$*, we say that* Π *is* (S, ϵ)*-secure if for all* $i \in \{0,1\}$ *and all non-uniform adversaries* $\mathcal{A}$ *of size* $S(\lambda)$ *and sufficiently large* λ*, we have* $\mathrm{Adv}(1^\lambda, \mathcal{A}, i) \leq \epsilon(\lambda)$*. We say that* Π *is:*

  – Computationally ϵ-secure *if it is* (S, ϵ)*-secure for all polynomials* S*.*

  – Computationally secure *if it is* $(S, 1/S)$*-secure for all polynomials* S*.*

We will also be interested in applying the evaluation algorithm on *all* inputs. Given a DPF $(\mathsf{Gen}, \mathsf{Eval}_0, \mathsf{Eval}_1)$, we denote by $\mathsf{EvalAll}_i$ an algorithm which computes Eval_i on every input x. Hence, $\mathsf{EvalAll}_i$ receives only a key k_i as input.

DPF efficieny measures. We will pay attention to the following efficiency measures of a DPF:

- The key sizes $|k_0|, |k_1|$.
- The running time of $\mathsf{Gen}, \mathsf{Eval}_0, \mathsf{Eval}_1$.

Small-domain and large-domain DPF. We say that a DPF is *small-domain* (resp., *large-domain*) if $\mathsf{Gen}, \mathsf{Eval}_0, \mathsf{Eval}_1$ have running time polynomial in N (resp., $\log N$) and their input length.

Next, we introduce our new notion of *programmable* DPF.

Definition 2 (PDPF)**.** *We say that a DPF* $\Pi = (\mathsf{Gen}, \mathsf{Eval}_0, \mathsf{Eval}_1)$ *is a* programmable DPF*, or PDPF for short, if* Gen *can be decomposed into a pair of algorithms* $(\mathsf{Gen}_0, \mathsf{Gen}_1)$ *with the following syntax:*

- $\mathsf{Gen}_0(1^\lambda, N, \hat{\mathbb{G}}, \hat{\mathbb{G}}') \to k_0$*: On input security parameter* λ*, domain size* N *and output group description* $\hat{\mathbb{G}}$*, returns a key* $k_0 = (k_*, N, \hat{\mathbb{G}}, \hat{\mathbb{G}}')$ *where* $k_* \in \{0,1\}^\lambda$*.*

- $\mathsf{Gen}_1(k_0, \hat{f}_{\alpha,\beta}) \to k_1$*: On input key* $k_0 = (k_*, N, \hat{\mathbb{G}}, \hat{\mathbb{G}}')$ *and point function description* $\hat{f}_{\alpha,\beta} = (N, \hat{\mathbb{G}}, \hat{\mathbb{G}}', \alpha, \beta)$*, returns a key* $k_1 \in \{0,1\}^*$*.*

By default, we assume Π *to be a* small-domain *DPF and require (without loss of generality) that* k_**, returned by* Gen_0 *as part of* k_0*, is a uniform random string, namely,* $k_* \xleftarrow{\$} \{0,1\}^\lambda$*.*

Since Gen_0 is a fixed algorithm that outputs a uniformly random k_0 of length λ, in our PDPF constructions we will omit the description of Gen_0. Finally, since our constructions will realize $\mathsf{EvalAll}$ at essentially the same cost as Eval, we will directly describe the $\mathsf{EvalAll}$ algorithm.

In Appendix A we define the *reusability* feature for DPFs discussed in the Introduction, and show an easy construction of reusable DPF from PDPF (and vice versa).

Simulation-based security. While for both DPF and PDPF we use a definition with indistinguishability-based security, there is an equivalent definition using simulation-based security [14]. There, the simulator is given "leakage" which is the description of the DPF function class, which in our case is specified by $N, \hat{\mathbb{G}}, \hat{\mathbb{G}}'$. Simulation takes place by simply generating a key for an arbitrary function in the function class.

2.2 Pseudorandom Generators and Functions

Below we recall the definitions of pseudorandom generators and pseudorandom functions.

Definition 3 (PRG [3, 45])**.** *A* pseudorandom generator *(PRG) with expansion $\ell(\lambda) > \lambda$ is a function $G : \{0,1\}^* \to \{0,1\}^*$, such that:*

- *For all $x \in \{0,1\}^\lambda$, $G(x) \in \{0,1\}^{\ell(\lambda)}$.*
- *G is computable in deterministic polynomial time.*

For advantage bound $\epsilon(\lambda)$ and polynomial circuit size bound $S(\lambda)$, we say that the PRG G is (S, ϵ)-secure if for any non-uniform adversary $\mathcal{A}$ of size $S(\lambda)$, its distinguishing advantage between $G(U_\lambda)$ and $U_{\ell(\lambda)}$ is at most $\epsilon(\lambda)$.

We simply call it a PRG if it is $(S, 1/S)$-secure for any polynomial S.

Definition 4 (PRF [31])**.** *A* pseudorandom function (PRF) *is a polynomial-time algorithm* Eval *with the following syntax:*

- $\mathsf{Eval}(k, M, N, x) \to y$*: On input key $k \in \{0,1\}^\lambda$, input domain size $M \in \mathbb{N}$, output domain size $N \in \mathbb{N}$, and point $x \in [M]$, the algorithm returns an output $y \in [N]$.*

For advantage bound $\epsilon(\lambda)$ and polynomial circuit size bound $S(\lambda)$, we say that the PRF Eval *is $(S(\lambda), \epsilon(\lambda))$-secure if for any non-uniform oracle adversary $\mathcal{A}$ of size $S(\lambda)$,*

$$\left| \Pr_{k \xleftarrow{\$} \{0,1\}^\lambda} \left[\mathcal{A}^{\mathsf{Eval}(k,M,N,\cdot)}(1^\lambda) = 1 \right] - \Pr_{f \xleftarrow{\$} [N]^{[M]}} \left[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1 \right] \right| \le \epsilon(\lambda),$$

where $[N]^{[M]}$ denotes the set of all functions from $[M]$ to $[N]$.

We simply call it a PRF if it is $(S, 1/S)$-secure for any polynomial S.

Definition 5 (PPRF)**.** *[7, 36, 17] A PRF* Eval *is a* puncturable PRF *(PPRF) if there exist additional polynomial time algorithms:*

- *$\mathsf{Punc}(k, M, N, x)$: On input key $k \in \{0,1\}^\lambda$, input domain size $M \in \mathbb{N}$, output domain size $N \in \mathbb{N}$, and point $x \in [M]$, and punctured point $x \in [M]$, the algorithm returns a punctured key $k_p \in \{0,1\}^*$.*
- *$\mathsf{PuncEval}(k_p, x)$: On input punctured key $k_p \in \{0,1\}^*$ and point $x \in [M]$, the algorithm returns a value $y \in [N]$.*

The algorithms $(\mathsf{Eval}, \mathsf{Punc}, \mathsf{PuncEval})$ *should satisfy the following additional requirements:*

- ***Correctness:*** *For every $\lambda, M, N \in \mathbb{N}$ and $x \neq x_p \in [M]$ it holds that*

$$\Pr\left[\begin{matrix} k \xleftarrow{\$} \{0,1\}^\lambda, \\ k_p \leftarrow \mathsf{Punc}(k, M, N, x_p) \end{matrix} : \mathsf{PuncEval}(k_p, x) = \mathsf{Eval}(k, M, N, x) \right] = 1.$$

- ***Security:*** *Consider the following experiment for some adversary $\mathcal{A}$:*
  - *The adversary gives challenge domain sizes and input*
  $$(M, N, x_p) \leftarrow \mathcal{A}(1^\lambda).$$
  - *The challenger draws $b \xleftarrow{\$} \{0,1\}$ and computes*
  $$k \xleftarrow{\$} \{0,1\}^\lambda, \quad k_p \leftarrow \mathit{Punc}(k, M, N, x_p),$$
  $$y_0 \xleftarrow{\$} [N], \quad y_1 \leftarrow \mathsf{Eval}(k, M, N, x_p).$$
  - *The adversary outputs a guess $b' \leftarrow \mathcal{A}(k_p, y_b)$.*

  For advantage bound $\epsilon(\lambda)$ and circuit size bound $S(\lambda)$, we say that the PPRF (Eval, *Punc*, *PuncEval*) *is $(S\lambda), \epsilon(\lambda))$-secure if for any non-uniform adversary $\mathcal{A}$ of size $S(\lambda)$ and sufficiently large λ, it holds in the above experiment that* $\Pr[b = b'] - 1/N \leq \epsilon(\lambda)$.

  We simply call it a PPRF if it is $(S, 1/S)$-secure for any polynomial S.

Note that Definition 5 does not specify the output value $\mathsf{PuncEval}(k_p, x_p)$. When not discussing *privately puncturable PRFs* (see Section 4.1 for more details) we will assume by default that the value of x_p is *known* from k_p, in which case PuncEval returns a default value on the punctured point, that is $\mathsf{PuncEval}(k_p, x_p) = \perp$.

Theorem 3 ((P)PRF from OWFs [7, 36, 17])**.** *If OWFs exist, there exists a PPRF. More concretely, given a black-box access to a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$, a PPRF, PPRF =* ($\mathsf{Eval}$, *Punc*, *PuncEval*)*, with input domain $[M]$ and output domain $[N]$, can be implemented with punctured key length $|k_p| = \lambda \log_2 M$, such that* Eval, *Punc*, *PuncEval make* $(\log_2(M/N)\log_2 N)/2\lambda$ *calls to G.*

Furthermore, if Eval *or PuncEval is computed on all points in $[M]$, it requires only* $((2M-1)\log_2 N)/2\lambda$ *calls to G.*

3 Small-Domain PDPF from One-Way Functions

In this section we construct small-domain PDPFs. We will first obtain a construction with inverse-polynomial security. Then, in Section 3.1, we will show how to amplify security and get negligible security error.

As was discussed in the introduction, our construction relies on analyzing the statistical distance between balls-and-bins experiments, where, after throwing M balls into N bins, we remove a single ball (randomly) from either bin i or bin j. The following lemma gives an exact expression for the statistical distance between these two distributions, and also provides an estimate which, numerically, is close up to a multiplicative factor of ≈ 0.564 (see Section 5).

Lemma 1. *For integers $M > N > 0$ and $i, j \in [N]$, let D_i and D_j be distributions over $\{1, \ldots, N, \perp\}^M \cup \{\mathit{fail}_i, \mathit{fail}_j\}$ of the locations of M balls independently and randomly thrown into N bins, such that we then change the position of a single ball, chosen randomly from bin i and bin j, respectively, to $\perp$ (this corresponds to the ball's "removal" from the bin). If there is no ball in bin i, then the output of D_i is fail_i. Then*

$$d(D_i, D_j) = \sum_{w=0}^{M} \binom{M}{w} \left(1 - \frac{2}{N}\right)^{M-w} \frac{\binom{w}{\lfloor w/2 \rfloor}}{N^w} \leq \sqrt{\frac{N}{M}} + 2^{-\Omega(M/N)}$$

Proof. In the following we prove the inequality part directly by analyzing the probabilistic experiment. This argument generalizes to the computational setting, hence we present it here as a warm-up. The equality part of the lemma, which is useful for empirically approximating the exact statistical distance, is proven Appendix B.

First, we want to prove an upper bound on $d(D_i, D_j)$. To this end, let $\tilde{D}$ be the distribution where after throwing M balls into N bins, the position of a random ball (regardless of bin) is changed to $\perp$. Denote by $r \xleftarrow{\$} [N]$ an independently chosen random element. We first claim that $d(D_i, D_j) \le 2d((D_r, r), (\tilde{D}, r))$. This holds because

$$d(D_i, D_j) \le d((D_i, i), (\tilde{D}, i)) + d((D_j, j), (\tilde{D}, j)) = 2d((D_r, r), (\tilde{D}, r)),$$

where the equality follows by symmetry. In addition, denote by $\tilde{r}$ the original bin of the removed ball in $\tilde{D}$. Because $\tilde{r}$ is independent of $\tilde{D}$ and uniformly random, we have that $(\tilde{D}, r) = (\tilde{D}, \tilde{r})$. Next, let H be distribution of configurations of throwing M balls into N bins without removing any ball. In addition, denote by ℓ_r the index of the removed ball in D_r, which is set as $\ell_r = \mathsf{undefined}$ whenever $D_r = \mathsf{fail}_r$. In addition, denote by $\tilde{\ell}$ the index of the removed ball in $\tilde{D}$. Then, because both (D_r, r) can be simulated knowing only (H, ℓ_r), and $(\tilde{D}, \tilde{r})$ can be simulated knowing only $(H, \tilde{\ell})$, *with the same randomized simulator*[7], we have that

$$\begin{aligned} d((D_r, r), (\tilde{D}, \tilde{r})) &\le d((H, \ell_r), (H, \tilde{\ell})) \\ &= \frac{1}{2} \sum_h \Pr[H = h] \sum_{\ell \ne \mathsf{undefined}} |\Pr[\ell_r = \ell | H = h] - \Pr[\tilde{\ell} = \ell | H = h]| \quad (*) \\ &+ \frac{1}{2} \sum_h \Pr[H = h] \left| \Pr[\ell_r = \mathsf{undefined} | H = h] - \Pr[\tilde{\ell} = \mathsf{undefined} | H = h] \right|. \quad (**) \end{aligned}$$

Next, we note that $\Pr[\tilde{\ell} = \mathsf{undefined} | H = h] = 0$. In addition, because r is chosen randomly, $\Pr[\ell_r = \mathsf{undefined} | H = h]$ is exactly the fraction of empty bins in h. Hence $(**)$ equals the expectation of the fraction of empty bins in H. The probability of each individual bin being empty is $(1 - 1/N)^M$, which by linearity of expectation equals the expected fraction of empty bins in H. Hence

$$(**) \le \left(1 - \frac{1}{N}\right)^M = 2^{-\Omega(M/N)}$$

To bound $(*)$, first let S_y be the set of balls in bin y in configuration H. Thus $(*)$ can be bounded as follows:

$$\begin{aligned} (*) &= \sum_h \Pr[H = h] \frac{1}{2} \sum_{y : |S_y| \ge 1} \sum_{\ell \in S_y} \left| \frac{1}{|S_y|} \cdot \frac{1}{N} - \frac{1}{M} \right| \\ &= \sum_h \Pr[H = h] \frac{1}{2} \sum_{y : |S_y| \ge 1} \left| \frac{1}{N} - \frac{|S_y|}{M} \right| \\ &\le \sum_h \Pr[H = h] \frac{1}{2} \sum_{y=1}^{N} \left| \frac{1}{N} - \frac{|S_y|}{M} \right| \\ &= \mathbb{E}_H \left[d\left(\left(\frac{1}{N}, \dots, \frac{1}{N} \right), \left(\frac{|S_1|}{M}, \dots, \frac{|S_N|}{M} \right) \right) \right], \end{aligned}$$

Our proof will be complete by demonstrating that $\mathbb{E}_H d\left(\left(\frac{1}{N}, \dots, \frac{1}{N} \right), \left(\frac{|S_1|}{M}, \dots, \frac{|S_N|}{M} \right) \right) \le \frac{1}{2}\sqrt{\frac{N}{M}}$. We have that $|S_y| \sim \mathit{Binomial}(M; 1/N)$. Therefore,

$$\left(\mathbb{E}_H \left[\left| |S_y| - \frac{M}{N} \right| \right] \right)^2 \le \mathbb{E}_H \left[\left| |S_y| - \frac{M}{N} \right|^2 \right] = \frac{M}{N} \left(1 - \frac{1}{N}\right) \le \frac{M}{N}.$$

[7]Whenever ℓ_r is not $\mathsf{undefined}$, simply remove this ball from H. Otherwise, pick a random *empty* bin r from H and output (fail_r, r). The same simulation is used for $\tilde{\ell}$, with the exception of $\tilde{\ell} = \mathsf{undefined}$ not happening.

Then, by the previous estimate and Cauchy-Schwarz,

$$\begin{aligned}\mathbb{E}_H\left[d\left(\left(\frac{1}{N},\ldots,\frac{1}{N}\right),\left(\frac{|S_1|}{M},\ldots,\frac{|S_N|}{M}\right)\right)\right] &= \frac{1}{2M}\sum_{y=1}^{N}\mathbb{E}_H\left[\left||S_y|-\frac{M}{N}\right|\right] \\ &\le \frac{1}{2M}\sum_{y=1}^{N}\sqrt{\frac{M}{N}} \\ &\le \frac{1}{2M}\left(\sqrt{\sum_{y=1}^{N}M}\right)\left(\sqrt{\sum_{y=1}^{N}\frac{1}{N}}\right) \\ &= \frac{1}{2}\sqrt{\frac{N}{M}}.\end{aligned}$$

□

Next, we prove the theorem below, which constructs a PDPF (Figure 1), restricted for the output group $\mathbb{Z}$ and to payloads $\beta \in \{0,1\}$. Later, we extend this PDPF in Theorem 5 to work over any finite Abelian group $\mathbb{G}$ and any payload $\beta \in \mathbb{G}$. The proof of the theorem below essentially mirrors that of Lemma 1 in the computational world by replacing the random configuration of M balls thrown into N bins by a pseudorandom one, using the truth table of a PPRF. Compared to Lemma 1, this yields an additive error term negligible in λ.

Theorem 4 (Small-domain PDPF with 1/poly privacy error)**.** *Suppose that there is a secure PPRF, $\mathsf{PPRF} = (\mathsf{PPRF.Eval}, \mathsf{PPRF.Punc}, \mathsf{PPRF.PuncEval})$, for input domain size M and output domain size N, with punctured key size $K_p(\lambda, M, N)$. In addition, let $G : \{0,1\}^\lambda \to [N+1] \times \{0,1\}^\lambda$ be a PRG. Then, the construction in Figure 1 is a small-domain computationally ϵ-secure PDPF,*

$$\epsilon(\lambda, M, N) = \sqrt{\frac{(N+1)}{M}} + 2^{-\Omega(M/N)} + \text{negl}(\lambda)$$

for point functions with output group $\mathbb{G} = \mathbb{Z}$, $\mathbb{G}' = \{0,1\}$, domain size N, and key size $|k_1| = K_p(\lambda, M, N+1)$. The number of invocations to PPRF in $\mathsf{Gen}_1, \mathsf{EvalAll}_0, \mathsf{EvalAll}_1$ is at most $O(M)$.

Proof of Theorem 4.

Efficiency: The key length assertion, as well as the estimate on the number of invocations to PPRF in all algorithms, follows by construction.

Correctness: Note that the algorithm Gen_1 does not account for the cases where $L = \emptyset$, however, in this case we can make Gen_1 to reveal k_{PPRF} and $\hat{f}_{\alpha,\beta}$ to still achieve perfect correctness at the cost of negligible privacy error. To this end, in similar fashion to Lemma 1, suppose that we output fail_α in the case of $\beta = 1$ or fail_{N+1} in the case of $\beta = 0$. Otherwise, correctness follows from the definition of PPRF.

Security: By the PRG security of G, we may view s, up to $\epsilon_{\mathsf{PRG}} = \text{negl}(\lambda)$ security error, as uniformly random and independent from all other values of PPRF, determined by k_{PPRF}.

Denote by D_i the output distribution of Gen_1 on $\alpha = i$ and $\beta = 1$. In addition, denote by D_{N+1} the output distribution of Gen_1 on $\beta = 0$. Furthermore, throughout the proof denote by $r \overset{\$}{\leftarrow} [N+1]$ an independently chosen random element, and by $\tilde{D}$ a modified output distribution of Gen_1, where $\ell \overset{\$}{\leftarrow} [M]$ instead of $\ell \overset{\$}{\leftarrow} L$. For all polynomial size adversaries $\mathcal{A}$, denote $\delta_i = \max_{\mathcal{A}} \text{Adv}[\mathcal{A}, (D_i, i), (\tilde{D}, i)]$.

```
$\mathsf{Gen}_1(k_0 = (k_*, N, \hat{\mathbb{G}}, \hat{\mathbb{G}'}), \hat{f}_{\alpha,\beta} = (N, \hat{\mathbb{G}}, \alpha, \beta))$:
    • Compute $(s, k_{\mathsf{PPRF}}) = G(k_*)$, where $s, k_{\mathsf{PPRF}} \in \{0,1\}^\lambda$.
    • If $\beta = 1$ then find all indices
          $L \leftarrow \{\ell \in [M] : \mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, M, N+1, \ell) + s = \alpha\}$ .
    • Else, if $\beta = 0$ then find all indices
          $L \leftarrow \{\ell \in [M] : \mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, M, N+1, \ell) + s = N+1\}$ .
    • Pick a random $\ell \in L$, compute $k_p \leftarrow \mathsf{PPRF.Punc}(k_{\mathsf{PPRF}}, M, N+1, \ell)$, and output $k_1 = (k_p, s)$.

$\mathsf{EvalAll}_0(k_0 = (k_{\mathsf{PPRF}}, N, \hat{\mathbb{G}}, \hat{\mathbb{G}'}))$:
    • Compute $(s, k_{\mathsf{PPRF}}) = G(k_*)$, where $s, k_{\mathsf{PPRF}} \in \{0,1\}^\lambda$.
    • For every $\alpha \in [N]$, simultaneously compute
          $Y_\alpha \leftarrow |\{\ell \in [M] : \mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, M, N+1, \ell) + s = \alpha\}|$ .
    • Output $Y = (Y_\alpha)_{\alpha \in [N]}$.

$\mathsf{EvalAll}_1(k_1 = (k_p, s))$:
    • For every $\alpha \in [N]$, simultaneously compute
          $Y_\alpha \leftarrow (-|\{\ell \in [M] : \mathsf{PPRF.PuncEval}(k_p, \ell) + s = \alpha\}|)$ .
    • Output $Y = (Y_\alpha)_{\alpha \in [N]}$.
```

Figure 1: Small-domain computationally 1/poly-secure PDPF for point functions with output group $\mathbb{G} = \mathbb{Z}$, payload set $\mathbb{G}' = \{0,1\}$, and domain size N. Here M is a parameter corresponding to the input space of the PPRF.

In similar fashion to Lemma 1, we want to show that $\epsilon \leq 2\delta_r$. In other words, we want to argue that $\epsilon \leq \delta_i + \delta_j = 2\delta_i = 2\delta_r$. Here, the first equality is not immediate. We will argue that $\delta_i \leq \delta_j$, which will imply the claim from the arbitrariness of i and j. Indeed, suppose that $\mathcal{A}_i$ is such that $\delta_i = \mathrm{Adv}[\mathcal{A}_i, (D_i, i), (\tilde{D}, i)]$. Let c_1, c_2 be the two components of either D_i or $\tilde{D}$. We will construct an adversary $\mathcal{A}_j$ as follows: On input $((c_1, c_2), j)$, output $\mathcal{A}_i((c_1, c_2 - (i-j)), i))$. Since (c_1, c_2) being distributed according to D_j implies that $(c_1, c_2 - (i-j))$ is distributed according to D_i, and (c_1, c_2) being distributed according to $\tilde{D}$ implies that $(c_1, c_2 - (i-j))$ is also distributed according to $\tilde{D}$, we have that $\epsilon \leq 2\delta_r$.

Next, we will bound $\delta_r \leq \sqrt{(N+1)/M} + \epsilon_{\mathsf{PRF}} + \epsilon_{\mathsf{PPRF}}$, where ϵ_{PRF} and ϵ_{PPRF} are the PRF and PPRF security errors, respectively. Denote by $\tilde{r}$ the distribution of $\mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, M, N+1, \ell)$, where $\ell \xleftarrow{\$} [M]$ (in similar fashion to how it was defined in the proof Lemma 1). Note that by PPRF security we have that $\max_{\mathcal{A}} \mathrm{Adv}[\mathcal{A}, (\tilde{D}, r), (\tilde{D}, \tilde{r})] \leq \epsilon_{\mathsf{PPRF}}$, implying that

$$\max_{\mathcal{A}} \mathrm{Adv}[\mathcal{A}, (\tilde{D}_r, r), (\tilde{D}, r)] \leq \max_{\mathcal{A}} \mathrm{Adv}[\mathcal{A}, (D_r, r), (\tilde{D}, \tilde{r})] + \epsilon_{\mathsf{PPRF}},$$

so, as in Lemma 1, we are left with bounding $\max_{\mathcal{A}} \mathrm{Adv}[\mathcal{A}, (D_r, r), (\tilde{D}, \tilde{r})]$. Recall that we view all values of PPRF as being drawn uniformly, and the punctured point ℓ is chosen depending on k_{PPRF} and s. Denote by ℓ_r the distribution where ℓ is chosen uniformly such that the output is r ($\ell_r = \mathsf{undefined}$ in the case a PPRF output r is absent), and by $\tilde{\ell} \xleftarrow{\$} [M]$ a uniformly random punctured point. Then, because both (D_r, r) can be simulated knowing only $(k_{\mathsf{PPRF}}, \ell_r)$, and $(\tilde{D}, \tilde{r})$ can be simulated knowing only $(k_{\mathsf{PPRF}}, \tilde{\ell})$, *with the same randomized simulator* (as in Lemma 1 where k_{PPRF} is replaced by H), it holds that

$$\max_{\mathcal{A}} \mathrm{Adv}[\mathcal{A}, (D_r, r), (\tilde{D}, \tilde{r})] \leq 2^{-\Omega(M/N)} + \mathbb{E}_{k_{\mathsf{PPRF}}} \left[d\left(\left(\frac{1}{N+1}, \ldots, \frac{1}{N+1} \right), \left(\frac{\tilde{S}_1}{M}, \ldots, \frac{\tilde{S}_{N+1}}{M} \right) \right) \right],$$

where $\tilde{S}_y = |\{\ell : \mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, M, N+1, \ell) = y\}|$. Next, we claim that by PRF security we have that if

$$\begin{aligned} \bar{\epsilon} &= \mathbb{E}_{k_{\mathsf{PPRF}}} \left[d\left(\left(\frac{1}{N+1}, \ldots, \frac{1}{N+1} \right), \left(\frac{\tilde{S}_1}{M}, \ldots, \frac{\tilde{S}_{N+1}}{M} \right) \right) \right] \\ &- \mathbb{E}_{H} \left[d\left(\left(\frac{1}{N+1}, \ldots, \frac{1}{N+1} \right), \left(\frac{S_1}{M}, \ldots, \frac{S_{N+1}}{M} \right) \right) \right] \end{aligned}$$

then $|\bar{\epsilon}| \leq \epsilon_{\mathsf{PRF}}$ where, as in the proof of Lemma 1, H is the distribution of configurations of throwing M balls into $N+1$ bins without removing any ball and S_y is the number of balls in bin y in configuration H. Indeed, this holds because there is an efficient adversary $\mathcal{A}^*$, that, given sets $T_1, \ldots, T_{N+1}$, outputs 1 with probability $d\left(\left(\frac{1}{N+1}, \ldots, \frac{1}{N+1}\right), \left(\frac{T_1}{M}, \ldots, \frac{T_{N+1}}{M}\right)\right)$. This follows since the statistical distance can be efficiently computed given the sets $T_1, \ldots, T_{N+1}$. Hence, we conclude that

$$\Pr_{k_{\mathsf{PPRF}}} [\mathcal{A}^*(\tilde{S}_1, \ldots, \tilde{S}_{N+1}) = 1] = \mathbb{E}_{k_{\mathsf{PPRF}}} \left[d\left(\left(\frac{1}{N+1}, \ldots, \frac{1}{N+1} \right), \left(\frac{\tilde{S}_1}{M}, \ldots, \frac{\tilde{S}_{N+1}}{M} \right) \right) \right],$$

$$\Pr_{H} [\mathcal{A}^*(S_1, \ldots, S_{N+1}) = 1] = \mathbb{E}_{H} \left[d\left(\left(\frac{1}{N+1}, \ldots, \frac{1}{N+1} \right), \left(\frac{S_1}{M}, \ldots, \frac{S_{N+1}}{M} \right) \right) \right],$$

which, by PRF security, implies that

$$\left| \Pr_{k_{\mathsf{PPRF}}} [\mathcal{A}^*(\tilde{S}_1, \ldots, \tilde{S}_{N+1}) = 1] - \Pr_{H} [\mathcal{A}^*(S_1, \ldots, S_{N+1}) = 1] \right| \leq \epsilon_{\mathsf{PRF}}.$$

As in the proof of Lemma 1, we can thus conclude that

$$\max_{\mathcal{A}} \mathrm{Adv}[\mathcal{A}, (D_r, r), (\tilde{D}, \tilde{r})] \leq \frac{1}{2} \sqrt{\frac{N+1}{M}} + 2^{-\Omega(M/N)} + \epsilon_{\mathsf{PRF}}$$

This gives an overall bound $\delta_r \leq \frac{1}{2}\sqrt{\frac{N+1}{M}} + 2^{-\Omega(M/N)} + \epsilon_{\mathsf{PRF}} + \epsilon_{\mathsf{PPRF}}$. By noting that ϵ_{PRF} and ϵ_{PPRF} are negligible in λ, we are done. ☐

Theorem 5 (Small-domain PDPF over any payload set $\mathbb{G}'$)**.** *Let P be a polynomial. If OWFs exists, there exists a small-domain computationally $1/p(\lambda)$-secure PDPF for point functions with any allowed payload set $\mathbb{G}'$, Abelian output group $\mathbb{G} \supseteq \mathbb{G}'$, domain size N, and key size $|k_1| = \text{poly}(\log|\mathbb{G}'|, \lambda, \log N)$.*

More concretely, the key size is given by $|k_1| = O(\log|\mathbb{G}'|\lambda(\log N + \log P(\lambda) + \log\log|\mathbb{G}'|))$.

Proof. It is possible to extend the construction in Theorem 4 to any $\beta \in \mathbb{Z}_q$, or, generally, a set $S \subseteq \mathbb{Z}$, by bit decomposition, which will incur a multiplicative factor of $\log|S|$ in privacy loss, computational cost, and key length. Furthermore, it is possible to extend it to any Abelian group $\mathbb{G}$ by decomposing $\mathbb{G}$ as a product of cyclic subgroups $\mathbb{G} = \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_\ell} \times \mathbb{Z}^r$, making the key k_1 a concatenation of the corresponding keys for each subgroup, which overall incurs a multiplicative factor of $\log|\mathbb{G}'|$ in privacy loss, computation cost, and key length. By choosing $M = O(N(P(\lambda)\log|\mathbb{G}'|)^2)$ and using Theorem 3 we are done. ☐

We finish this section with an optimization to Theorem 4.

Proposition 4 (Lazy Gen computation)**.** *When instantiated with a PPRF from Theorem 3, the computation of Gen_1 in Figure 1 can be done in just $((N+1)\log_2 M)/\lambda$ calls to a PRG, at the expense of an additional $2^{-(N+1)}$ error in correctness or privacy.*

Proof. Instead of having Gen_1 compute the entire set L and picking $\ell \in L$ at random, it is sufficient to keep trying values $\ell \in [M]$ at random until one is found such that $\mathsf{PPRF.Eval}(k_*, M, N+1, \ell) + s$ takes the correct value. This has a $1/(N+1)$ probability of success. By making T queries, the chance of failure is $1/(N+1)^T$. If we pick $T = (N+1)/\log_2(N+1)$, the failure chance becomes $2^{-(N+1)}$, which we can attribute to either correctness or privacy. Since each PPRF evaluation takes $(\log_2 M \log_2(N+1))/\lambda$ calls to the PRG, we are done. ☐

3.1 Security Amplification

To amplify security we rely on Locally Decodable Codes (LDC). Theorem 4 gives us a PDPF with 1/poly leakage of α, which as we argue later (see Lemma 3), is no worse than α leaking completely with probability 1/poly, and staying (computationally) hidden otherwise. By utilizing a locally decodable code with additive decoding we can essentially secret share α into shares $\alpha_1, \ldots, \alpha_q$ which are λ-wise independent. Since every α_i leaks independently with small probability, by using a Chernoff bound, α leaks with negligible probability.

To describe the main idea of the security amplification construction (Figure 2) in more detail, note first that $f_\alpha(x) = \langle e_x, TT(f_\alpha)\rangle$, where e_x is a unit vector with 1 at index x, and $TT(f_\alpha)$ is the truth table of a point function f_α (also a unit vector). Now, we utilize a q-query LDC C with additive reconstruction and choose $\alpha_1, \ldots, \alpha_q$ to be the queries to C for coordinate α, which by the additive decoding of C yields

$$\langle C(e_x), TT(f_{\alpha_1}) + \ldots + TT(f_{\alpha_q})\rangle = \langle e_x, TT(f_\alpha)\rangle = f_\alpha(x).$$

Next, using the additive reconstruction of the PDPF, implying $TT(f_{\alpha_j}) = TT(f^0_{\alpha_j}) + TT(f^1_{\alpha_j})$, $j = 1, \ldots, q$, each server $i = 0, 1$ can locally compute $z_i = \langle C(e_x), TT(f^i_{\alpha_1}) + \ldots + TT(f^i_{\alpha_q})\rangle$ using $\mathsf{EvalAll}$ of each of the q PDPF keys, such that $z_0 + z_1 = f_\alpha(x)$ (hence yielding a PDPF). Here, the offline server will receive a *single* offline key, which it can expand to q offline keys using a PRF, while the online server will receive the q matching online keys.

The following lemma provides the locally decodable code (LDC) with the parameters we require (c.f. [18, Section 4]).

Lemma 2. *Fix integers $\sigma, w, r, N > 0$, such that $N \leq \binom{r+w}{r}$ and let p be a prime. There exist a deterministic mapping $C : \mathbb{Z}_p^N \to \mathbb{Z}_p^L$ and a randomized mapping $d : [N] \to [L]^q$, $L, q \in \mathbb{N}$, such that for every $z \in \mathbb{Z}_p^N$ and $\alpha \in [N]$ it holds that*

$$\Pr\left[\Delta \leftarrow d(\alpha) : \sum_{\ell=1}^{q} C(z)_{\Delta_\ell} = z_\alpha\right] = 1.$$

Moreover, the following properties hold:

1. *$q = O(\sigma^2 r)$ and $L = O(p^{w+1}\sigma^{w+1}r^{w+1})$.*
2. *C, d are computable in polynomial time.*
3. *For every $\alpha \in [N]$, the random variables $\Delta_1, \ldots, \Delta_q$ are σ-wise independent.*

Intuitively, C corresponds to the LDC encoder taking N symbols to $L > N$, the randomized mapping d determines the set of q queried symbols of the codeword given a target index $\alpha \in [N]$ of the "message" vector $z \in \mathbb{Z}_p^N$, and the decoding procedure is simply the sum of the queried symbols $\sum_{\ell=1}^{q} C(z)_{\Delta_\ell} = z_\alpha$. For example, these requirements can be met by a form of Reed-Muller code, where the distribution of queried points $\Delta \leftarrow d(\alpha)$ corresponds to random σ-degree polynomial evaluations through the desired point (namely, Shamir secret sharing of α).

Proof of Lemma 2. Choose $q = (\sigma+1)(r\sigma+1)$ and let $\mathbb{F} = \mathbb{F}_{p^c}$, $c \geq 1$, be the smallest finite field such that $|\mathbb{F}| > r\sigma + 1$, hence $|\mathbb{F}| = O(pr\sigma)$. Fix a canonical set of N points in $\mathbb{F}^w$ in general position, denoted by x_α for $\alpha \in [N]$. We choose $L = |\mathbb{F}|^{w+1} = O(p^{w+1}\sigma^{w+1}r^{w+1})$.

Encoding. To encode $z = (z_1, \ldots, z_N) \in \mathbb{Z}_p^N$, define the corresponding w-variable r-degree polynomial $P_z \in \mathbb{F}[X_1, \ldots, X_q]$ as the low-degree interpolation of evaluations $P_z(x_\alpha) = z_\alpha$, $\alpha \in [N]$. Let $\varphi : \mathbb{F} \to \mathbb{Z}_p$ be an additive homomorphism. The codeword $C(z)$ at point $(\rho, x) \in \mathbb{F} \times \mathbb{F}^w$ (we can alternatively index by elements of $[L]$) takes the value

$$C(z)_{(\rho,x)} = \varphi(\rho \cdot P_z(x)).$$

Decoding. To decode z_α, $\alpha \in [N]$, from $C(z)$, sample a random degree-σ parametric curve $\{\gamma(s) : s \in \mathbb{F}\} \subseteq \mathbb{F}^w$ that intersects x_α at $s = 0$, that is $\gamma(0) = x_\alpha$. Select a random sequence $(s_0, \ldots, s_{r\sigma}) \in \mathbb{F}^{r\sigma+1}$ of distinct *nonzero* parameter values, and let $c_0, \ldots, c_{r\sigma}$ be Lagrange coefficients corresponding to a choice of parameter values. The values $(d(\alpha))_{(\sigma+1)\ell}, \ldots, (d(\alpha))_{(\sigma+1)(\ell-1)+1} \in \mathbb{F} \times \mathbb{F}^w$, $\alpha \in [N], \ell \in [r\sigma+1]$ are defined to be

$$(d(\alpha))_{(\sigma+1)\ell} = (u_\ell^{\sigma+1}, \gamma(s_\ell)), \ldots, (d(\alpha))_{(\sigma+1)(\ell-1)+1} = (u_\ell^1, \gamma(s_\ell))$$

where $u_\ell^1, \ldots, u_\ell^{\sigma+1} \in \mathbb{F}$ satisfy $\sum_{j=1}^{\sigma+1} u_\ell^j = c_\ell$, and are otherwise random.

Correctness. It holds that

$$\begin{aligned}
\sum_{\ell=1}^{q} C(z)_{(d(\alpha))_\ell} &= \sum_{\ell=1}^{r\sigma+1}\sum_{j=1}^{\sigma+1} C(z)_{(\gamma(s_\ell), u_\ell^j)} \\
&= \varphi\left(\sum_{\ell=1}^{r\sigma+1}\sum_{j=1}^{\sigma+1} u_\ell^j \cdot P_z(\gamma(s_\ell))\right) \\
&= \varphi\left(\sum_{\ell=1}^{r\sigma+1} c_\ell \cdot P_z(\gamma(s_\ell))\right) \\
&= P_z(\gamma(0)) = P_z(x_\alpha) = z_\alpha
\end{aligned}$$

σ-wise independence. By construction, $\{\gamma(s_\ell)\}_\ell$ and $\{u_\ell^j\}_{\ell,j}$ are cross independent. In addition, both $\{\gamma(s_\ell)\}_\ell$ and $\{u_\ell^j\}_{\ell,j}$ are σ-wise independent. Hence $\{d(\alpha)_\ell\}_\ell$ are also σ-wise independent. □

Next, we show that any small-domain computationally 1/poly-secure PDPF can be transformed into a small-domain PDPF.

Notation: Let $C : \mathbb{Z}_p^N \to \mathbb{Z}_p^L$ and $d : [N] \to [L]^q$ be the mappings from Lemma 2. In addition, let $(\mathsf{PDPF.Gen}_1, \mathsf{PDPF.EvalAll}_0, \mathsf{PDPF.EvalAll}_1)$ be a small-domain computationally $O(1/q)$-secure PDPF for point functions with Abelian output group $\mathbb{Z}_p$, domain size L, and let $\mathsf{PRF.Eval}$ be a PRF.

$\mathsf{Gen}_1(k_0 = (k_*, N, \hat{\mathbb{G}}), \hat{f}_{\alpha,\beta} = (N, \hat{\mathbb{G}} = \widehat{\mathbb{Z}_p}, \alpha, \beta))$:

- Compute $\Delta \leftarrow d(\alpha)$.
- For $\ell = 1, \ldots, q$ let $k_*^\ell = \mathsf{PRF.Eval}(k_*, q, \lambda, \ell)$, $k_0^\ell = (k_*^\ell, L, \widehat{\mathbb{Z}_p})$, and
$$k_1^\ell \leftarrow \mathsf{PDPF.Gen}_1(k_0^\ell, (L, \widehat{\mathbb{Z}_p}, \Delta_\ell, \beta)).$$
- Output $k_1 = (k_1^1 \ldots, k_1^q)$.

$\mathsf{Eval}_0(k_0 = (k_*, N, \hat{\mathbb{G}} = \widehat{\mathbb{Z}_p}), x)$:

- For $\ell = 1, \ldots, q$ let $k_*^\ell = \mathsf{PRF.Eval}(k_*, q, \lambda, \ell)$ and $k_0^\ell = (k_*^\ell, L, \widehat{\mathbb{Z}_p})$.
- Compute and output
$$\left\langle C(e_x), \sum_{\ell=1}^{q} \mathsf{PDPF.EvalAll}_0(k_0^\ell) \right\rangle,$$
where $e_x \in \{0,1\}^L$ is a unit vector with 1 at index x.

$\mathsf{Eval}_1(k_1 = (k_1^1 \ldots, k_1^q), x)$:

- Compute and output
$$\left\langle C(e_x), \sum_{\ell=1}^{q} \mathsf{PDPF.EvalAll}_1(k_1^\ell) \right\rangle,$$
where $e_x \in \{0,1\}^L$ is a unit vector with 1 at index x.

Figure 2: Security amplification via LDC

Theorem 6. *Fix integers σ, w, r, N,be integers such that $N \leq \binom{r+w}{r}$, and let p be a prime. Furthermore, let $L = L(w, \sigma, N), q = q(w, \sigma, N)$ be as in Lemma 2. Suppose there exists a small-domain computationally $O(1/q)$-secure PDPF for point functions with Abelian output group $\mathbb{Z}_p$, domain size L, and key size $|k_1| = K$. Then, the construction in Figure 2 gives a $(2^{-\Omega(\sigma)} + \mathrm{negl}(\lambda))$-secure PDPF for point functions with Abelian output group $\mathbb{Z}_p$, domain size N, and key size $|k_1| = q \cdot K$, and where the running time of Gen_1 and $\mathsf{EvalAll}$ is $L \cdot \mathrm{poly}(q, \lambda, N, \log p)$.*

Corollary 2. *Let $p = \mathrm{poly}\log N$ be a prime and suppose that $\lambda \leq N$. If OWFs exist, there exists a small-domain PDPF for point functions with Abelian output group $\mathbb{Z}_p$, domain size N, and key size $|k_1| = \mathrm{poly}(\log N)$.*

Proof. Let $w = \log N / \log\log N$, $r = \log^2 N$, $\sigma = \log^2 \lambda$. Then $q = \mathrm{poly}\log N$ and
$$L = O(p^{w+1}\sigma^{w+1}r^{w+1}) = (\mathrm{poly}\log N)^{(\log N + \log\log \lambda)/\log\log N} = \mathrm{poly}(N),$$
where the last equality follows because $\lambda \leq N$. Furthermore, $2^{-\Omega(\sigma)} = 2^{-\Omega(\log^2 \lambda)} = \mathrm{negl}(\lambda)$. Thus, the PDPF is computationally secure, requires $\mathrm{poly}(N)$ computational cost for Gen and $\mathsf{EvalAll}$, and has key size $\mathrm{poly}(\log N)$. □

Remark 1. *Via CRT Corollary 2 can be generalized to handle any smooth integer characteristic. By introducing a small correctness error and converting it to privacy error Corollary 2 can be generalized to handle any Abelian group.*

Theorem 5 and Corollary 2 have the downside that their key length grows multiplicatively with $\log|\mathbb{G}|$. We show in Section 4.4 that this can be reduced to an additive term whenever $\log|\mathbb{G}| \gg \lambda$, at the cost of losing programmability, which still has the benefit of a DPF with one short $(\lambda + \log|\mathbb{G}|)$-length key.

3.1.1 Proof of Theorem 6

Before proving Theorem 6, we formally capture in Lemma 3 in which sense ϵ-leaking a secret is not worse than completely leaking the secret with probability ϵ, and completely hiding it with probability $1-\epsilon$.

Lemma 3. *Let $L = L(\lambda)$ be a function of the security parameter λ and let $\rho_1 = \rho_{1,\lambda} : [L] \to \{0,1\}^*$ be an efficiently computable randomized function. In addition, let $\delta = \delta(\lambda)$ be a function such that for every polynomial distingisher $\mathcal{A}$ it holds for sufficiently large λ that $\max_{i\in[L]} \mathrm{Adv}[\mathcal{A}, \rho_1(i), \rho_1(0)] \le \delta(\lambda)$.*

Then, there exist randomized functions, $\tau_\delta = \tau_{\delta,\lambda} : [L] \to [L] \cup \{\perp\}$, such that for every $i \in L$

$$\Pr[\tau_\delta(i) = i] \le \delta, \quad \Pr[\tau_\delta(i) = \perp] = 1 - \Pr[\tau_\delta(i) = i],$$

and $\rho_2 = \rho_{2,\lambda} : [L] \cup \{\perp\} \to \{0,1\}^*$, which is efficiently computable, such that for every $i \in [L]$

$$\rho_1(i) \overset{c}{\approx} \rho_2(\tau_\delta(i)),$$

where τ_δ and ρ_2 depend on the same *random coins.*

For this we require the hardcore lemma result of [38].

Lemma 4 ([38, Theorem 4])**.** *Let $F_1, F_2 : [R] \to \{0,1\}^*$ be functions, and let $\delta, \epsilon \in (0,1)$ and $T > 0$ be given. If for all distinguishers $\mathcal{A}$ with size T we have*

$$\mathrm{Adv}[\mathcal{A}, F_1(r), F_2(r)] \le \delta$$

whenever $r \xleftarrow{\$} [R]$, then there exists a set $Q \subseteq [R]$ with $|Q| \ge (1-\delta)R$ such that

$$\mathrm{Adv}[\mathcal{A}', F_1(r'), F_2(r')] \le \epsilon,$$

$r' \xleftarrow{\$} Q$, for all distinguishers $\mathcal{A}'$ with size $T' = \frac{T\epsilon^2}{128(2\log R)+1}$.

We are now ready to prove Lemma 3.

Proof of Lemma 3. In the proof we will view ρ_1 as a *deterministic* function of its input and randomness, namely, $\rho_1 : [L] \times [R] \to \{0,1\}^*$. Because ρ_1 is efficiently computable, $R = R(\lambda)$ is polynomial in λ. For every $\ell \in [L] \cup \{\perp\}$ and $r \in [R]$ define

$$\rho_2(\ell; r) = \begin{cases} \rho_1(\ell; r), & \ell \ne \perp \\ \rho_1(0; r), & \ell = \perp. \end{cases}$$

Let $\mathcal{A}$ be a distinguisher of size $T = T(\lambda)$, where T is a polynomial to be set later. In addition, let $\epsilon = \epsilon(\lambda)$ also be set later. Assuming λ is sufficiently large, applying Lemma 4 to $F_1(\cdot) = \rho_1(\ell;\cdot)$ and $F_2 = \rho_1(0;\cdot)$ implies the existence of a set $Q_\ell \subseteq [R]$, $|Q_\ell| \ge (1-\delta)R$ such that

$$\mathrm{Adv}[\mathcal{A}', \rho_1(\ell; r'), \rho_1(0; r')] \le \epsilon,$$

$r' \xleftarrow{\$} Q_\ell$, for all distinguishers $\mathcal{A}'$ of size $T' = (T\epsilon^2)/(128(2\log R)+1)$. Next, define $\tau_\delta : [L] \times [R] \to \{0,1\}^*$ for $\ell \in [L]$ and $r \in [R]$ as follows

$$\tau_\delta(\ell, r) = \begin{cases} \ell, & r \notin Q_\ell \\ \perp, & r \in Q_\ell. \end{cases}$$

Denote by E_ℓ the event that $\mathcal{A}(\rho_1(\ell;r)) = 1$. Hence, denoting $r' \xleftarrow{\$} Q_\ell$, we have for every distinguisher $\mathcal{A}'$ of size T' and $\ell \in [L]$ that

$$\begin{aligned}
\mathrm{Adv}[\mathcal{A}', \rho_1(\ell;r), \rho_2(\tau_\delta(\ell;r);r)] &\leq \Pr[r \in Q_\ell] \cdot |\Pr[E_\ell | r \in Q_\ell] - \Pr[E_0 | r \in Q_\ell]| \\
&+ \Pr[r \notin Q_\ell] \cdot |\Pr[E_\ell | r \notin Q_\ell] - \Pr[E_\ell | r \notin Q_\ell]| \\
&= \Pr[r \in Q_\ell] \mathrm{Adv}[\mathcal{A}', \rho_1(\ell;r'), \rho_1(0;r')] \\
&+ \Pr[r \notin Q_\ell] \cdot 0 \\
&\leq \mathrm{Adv}[\mathcal{A}', \rho_1(\ell;r'), \rho_1(0;r')] \leq \epsilon,
\end{aligned}$$

where the equality follows because $\mathrm{Adv}[\mathcal{A}, \rho_1(\ell;r'), \rho_1(0;r')] = |\Pr[E_\ell | r \in Q_\ell] - \Pr[E_0 | r \in Q_\ell]|$ holds by definition. Hence, for all adversaries $\mathcal{A}'$ of size T' we have $\mathrm{Adv}[\mathcal{A}', \rho_1(\ell;r), \rho_2(\tau_\delta(\ell;r);r)] \leq \epsilon$.

To finish the proof, let $P_1(\lambda), P_2(\lambda)$ be two arbitrary polynomials. Then, by setting $\epsilon(\lambda) = 1/P_1(\lambda)$ and $T(\lambda) = P_2(\lambda) \cdot (128(2\log R(\lambda)+1)+1)/\epsilon^2(\lambda)$ we conclude that for all adversaries $\mathcal{A}'$ of size $T'(\lambda) = P_2(\lambda)$ it holds that

$$\mathrm{Adv}[\mathcal{A}', \rho_1(\ell;r), \rho_2(\tau_\delta(\ell;r);r)] \leq \frac{1}{P_1(\lambda)}.$$

This is the definition of computational indistinguishability, hence $\rho_1(\ell) \overset{c}{\approx} \rho_2(\tau_\delta(\ell))$.

□

We are ready to prove Theorem 6, where the security will follow by applying Lemma 3 to argue that each of the q instances, which are σ-wise independent, independently completely leaks with small probability, and thus the chance of at least σ of them completely leaking is negligible.

Proof of Theorem 6.
Efficiency: The key size of k_1 follows by construction. The running time of Gen_1 is $\mathrm{poly}(\lambda, N, \log p) \cdot q$. The running time of Eval_i, $i = 0, 1$, is $L \cdot q \cdot \mathrm{poly}(\lambda, N, \log p)$.

Correctness: By correctness of the PDPF it holds that

$$\mathsf{PDPF.EvalAll}_0(k_0^\ell) + \mathsf{PDPF.EvalAll}_1(k_1^\ell) = \beta e_{\Delta_\ell},$$

where $e_{\Delta_\ell} \in \{0,1\}^L$ is a unit vector with 1 at index Δ_ℓ, and so

$$\begin{aligned}
\left\langle C(e_x), \sum_{\ell=1}^{q} \beta e_{\Delta_\ell} \right\rangle &= \sum_{\ell=1}^{q} \beta C(e_x)_{\Delta_\ell} \\
&= \begin{cases} \beta, & x = \alpha \\ 0, & x \neq \alpha \end{cases}
\end{aligned}$$

where the last equality follows by Lemma 2 as $\Delta \leftarrow d(\alpha)$.

Security: To show security we need to show that for every α and α', the keys $k_1 = (k_1^q, \ldots, k_1^q)$ and $k_1' = ((k_1^1)', \ldots, (k_1^q)')$, obtained by running Gen_1 on α and α', respectively, are computationally indistinguishable. For $\ell = 1, \ldots, q$ let ρ_1^ℓ be the function which maps a query $\Delta_\ell \in [L]$ to $\mathsf{PDPF.Gen}_1(k_0^\ell, (L, \widehat{\mathbb{Z}_p}, \Delta_\ell, \beta))$. Then, by Lemma 3, there is a randomized function ρ_2^ℓ and real number δ_ℓ such that, for every $i \in [L]$, $\rho_1^\ell(i)$ is computationally indistinguishable from $\rho_2^\ell(\tau_{\delta_\ell}(i))$, where $\delta_\ell = O(1/q)$ follows from the assumption that for every $i \in [L]$ we have $\mathrm{Adv}[\mathcal{A}, \rho_1^\ell(i), \rho_1^\ell(0)] = O(1/q)$ for all polynomial adversaries $\mathcal{A}$, because PDPF is $O(1/q)$-secure. Denoting $r_\ell = \tau_{\delta_\ell}(d(\alpha)_\ell)$ and $r_\ell' = \tau_{\delta_\ell}(d(\alpha')_\ell)$ we conclude that

$$\begin{aligned}
\max_{\mathcal{A}} \mathrm{Adv}[\mathcal{A}, k_1, k_1'] &\leq \max_{\mathcal{A}} \mathrm{Adv}\left[\mathcal{A}, (\rho_2^1(r_1), \ldots, \rho_2^q(r_q)), (\rho_2^1(r_1'), \ldots, \rho_2^q(r_q'))\right] + \mathrm{negl}(\lambda) \\
&\leq d\left((\rho_2^1(r_1), \ldots, \rho_2^q(r_q)), (\rho_2^1(r_1'), \ldots, \rho_2^q(r_q'))\right) + \mathrm{negl}(\lambda) \\
&\leq d\left((r_1, \ldots, r_q), (r_1', \ldots, r_q')\right) + \mathrm{negl}(\lambda)
\end{aligned}$$

Let X_i be the indicator to the event that $r_\ell = d(\alpha)_\ell$, and define X'_i similarly with respect to α' instead of α. Let E be the event that $\sum_{i=1}^{q} X_i \leq \sigma$, and define E' similarly. Then, because in the hybrid world all values $\{\mathsf{PRF.Eval}(k_*, q, \lambda, \ell)\}_{\ell=1}^{q}$ are uniformly random and independent, we also have that $\{X_i, X'_i\}_{i=1}^{q}$ are all independent, and so by Chernoff's inequality and the fact that $\mathbb{E}[X_i] = \mathbb{E}[X'_i] = O(1/q)$, we conclude that

$$\begin{aligned}\max_{\mathcal{A}} \mathrm{Adv}\left[\mathcal{A}, k_1, k'_1\right] &\leq d\left((r_1, \ldots, r_q)|_E, (r'_1, \ldots, r'_q)|_{E'}\right) + \Pr_{k_1}[\neg E] + \Pr_{k'_1}[\neg E'] + \mathrm{negl}(\lambda) \\ &= \Pr_{k_1}\left[\sum_{i=1}^{q} X_i > \sigma\right] + \Pr_{k'_1}\left[\sum_{i=1}^{q} X'_i > \sigma\right] + \mathrm{negl}(\lambda) \\ &\leq 2 \cdot 2^{-\Omega(\sigma)} + \mathrm{negl}(\lambda),\end{aligned}$$

where the equality follows because $(\alpha_1, \ldots, \alpha_q)$ and $(\alpha'_1, \ldots, \alpha'_q)$ are σ-wise independent and so

$$d\left((r_1, \ldots, r_q)|_E, (r'_1, \ldots, r'_q)|_{E'}\right) = 0.$$

□

4 Applications

In this section, we present three applications of our programmable DPF construction and associated techniques: (1) Privately Puncturable PRFs (on polynomial-size domains) from the minimal assumption of one-way functions; (2) (Standard) Distributed Point Functions that admit particularly efficient secure distributed key generation protocols, namely the first to achieve constant round complexity while making only black-box use of oblivious transfer and a pseudorandom generator; and (3) A new application regime of *trusted-offline* pseudorandom correlation generators. We discuss each in turn within the following subsections.

4.1 Privately Puncturable PRFs

Our programmable DPF construction makes use of puncturable pseudorandom functions (PRFs); namely, PRFs supporting generation of punctured keys that enable evaluation of the PRF on all but a single punctured input x^*. Puncturable PRFs are lightweight objects, with simple constructions known from one-way functions [7, 36, 17] (for example, in a GGM-tree PRF on n-bit inputs, simply give the n co-path PRG evaluations). However, all such known simple constructions inherently *reveal* the identity of the punctured input x^*.

Interestingly, if one wishes to obtain the same functionality, while *hiding* the identity of x^*, the corresponding object becomes much more challenging to obtain. Such notion is known as a *privately* puncturable PRF [6]. In contrast to the simple puncturable PRF constructions, despite significant effort, the only known instantiations of privately puncturable PRFs make use of heavy public-key cryptography machinery, and rely on structured public-key assumptions such as the Learning with Errors assumption or multi-linear maps [5, 20, 19, 42].

This challenging state of affairs remains the situation even for the case where the domain of the PRF is of feasible size. Indeed, there is no clear way "scale down" the constructions from above to a polynomial-size domain in a way that lessens the computational assumption, without reverting to trivial constructions where the key size grows to the entire truth table. Placing a requirement that the key size be sublinear in the domain size (or polylogarithmic, to more closely match the large-domain case), then the resulting notion falls in the same state of knowledge as in the general case: necessitating one-way functions, but only known to be achievable from the heavy public-key cryptography as above.

We observe that our notion of programmable DPF in fact directly *implies* privately puncturable PRFs with the same parameters. In turn, we provide the first construction of privately puncturable PRFs (on polynomial-size domains) from the minimal assumption of *one-way functions*.

We next present the definition of privately puncturable PRFs, together with our new feasibility result. We adapt the definition to mirror our PRF syntax, where Eval and Punc explicitly take the input domain size $M \in \mathbb{N}$ as input. For simplicity, we focus on the case of output space $\mathbb{Z}_2$, and thus omit output domain size from the syntax (we can, however, support more general output spaces as in Corollary 2). As with essentially all known constructions of privately constrained PRFs, we consider a setting of selective security, with security against 1 key query. We remark that in this setting, it was shown that indistinguishability-based and simulation-based definitions are equivalent [20].

Definition 6 (Privately Puncturable PRF (1-Key, Selective Security))**.** *A puncturable PRF* (Gen, Punc, Eval, $\mathit{PuncEval}$) *as in Definition 5 is a (selectively secure, 1-key)* privately puncturable PRF *family if for every non-uniform polynomial-time stateful adversary $\mathcal{A}$, there exists a polynomial-time simulator Sim such that the following are computationally indistinguishable:*

$$\{\mathsf{REAL}_{\mathcal{A}}(1^\lambda)\}_{\lambda\in\mathbb{N}} \overset{c}{\cong} \{\mathsf{IDEAL}_{\mathcal{A},\mathit{Sim}}(1^\lambda)\}_{\lambda\in\mathbb{N}},$$

where the real and ideal experiments are defined as follows:

Experiment $\mathsf{REAL}_{\mathcal{A}}(1^\lambda)$	*Experiment* $\mathsf{IDEAL}_{\mathcal{A},\mathit{Sim}}(1^\lambda)$
$x^* \leftarrow \mathcal{A}(1^\lambda)$	$x^* \leftarrow \mathcal{A}(1^\lambda)$
$k \leftarrow \mathsf{Gen}(1^\lambda)$	$k^* \leftarrow \mathit{Sim}(1^\lambda)$
$k^* \leftarrow \mathit{Punc}(k, M, x^*)$	$b \leftarrow \mathcal{A}(k^*)$*; Output* b
$b \leftarrow \mathcal{A}(k^*)$*; Output* b	

Intuitively, this notion of privately puncturable PRFs are directly implied by programmable DPFs, by taking the master PRF key to be the first-server DPF share, and generating a punctured key at x^* by computing a second-server DPF share for the function $f_{\alpha,\beta}$ with $\alpha = x^*$ and $\beta \leftarrow \{0,1\}$ selected at random.

Proposition 5 (Small-Domain Privately Puncturable PRF from OWF)**.** *Assume the existence of a length-doubling PRG (implied by OWF). Then there exists a (selectively secure, 1-key) privately puncturable PRF* (Gen, Punc, Eval, $\mathit{PuncEval}$)*, with the following complexity properties:*

- $\mathsf{Gen}(1^\lambda)$ *outputs a master PRF key of size λ bits; $\mathit{PuncEval}$ on domain size M outputs a punctured key of size* $\mathrm{poly}(\lambda, \log(M))$ *bits.*
- *The runtime of Punc and $\mathit{PuncEval}$ on domain size M consists of $O(N)$ PRG evaluations. In particular, for polynomial-size domain $M = M(\lambda)$, then Punc and $\mathit{PuncEval}$ each run in probabilistic polynomial time.*

Proof. By Corollary 2, taking parameter $w = \log M$ (note that the PRF input domain size M plays the role of the PDPF input domain size N), we know that the existence of a length-doubling PRG implies a PDPF construction $(\mathsf{Gen}, \mathsf{Eval}_0, \mathsf{Eval}_1)$ for output space $\mathbb{Z}_2$ with key size $\mathrm{poly}(\lambda, \log M)$, and where the runtime of Gen_1 and Eval_1 on domain size M consists of $O(M)$ PRG evaluations. Consider the following privately puncturable PRF construction:

- $\mathsf{Gen}(1^\lambda)$: Execute Gen_0 of the PDPF.
- $\mathsf{Punc}(k, M, x^*)$: Execute $\mathsf{Gen}_1(k, \hat{f}_{\alpha,\beta})$, where $\alpha = x^* \in [M]$, and $\beta \leftarrow \mathbb{Z}_2$ is selected at random.
- $\mathsf{Eval}(k, M, x)$ and $\mathsf{PuncEval}(k^*, x)$: Execute Eval_0 and Eval_1, respectively.

Correctness of the construction follows immediately from correctness of the PDPF. Regarding privacy of the punctured point given a punctured key, consider the following simulator. $\mathsf{Sim}(1^\lambda)$ runs $k_0 \leftarrow \mathsf{Gen}_0(1^\lambda, M, \hat{\mathbb{Z}}_2)$, followed by $k^* \leftarrow \mathsf{Gen}_1(k_0, \hat{f}_{0,0})$, for a *fixed* point function $\hat{f}_{0,0}$. By the security of the PDPF, it holds that this simulated punctured key (i.e., DPF key for $\hat{f}_{0,0}$ is computationally indistinguishable from the true punctured key (i.e., DPF key for $\hat{f}_{\alpha,\beta}$), as desired. Note further that given the punctured input x^*, pseudorandomness on input x^* additionally follows by the PDPF key security, as the true punctured key (DPF key on $\hat{f}_{\alpha,\beta}$) cannot be distinguished from an alternative key for $\hat{f}_{\alpha,\beta'}$ with independently sampled offset $\beta' \leftarrow \mathbb{Z}_2$. $\square$

Remark 2 (Privately Puncturable PRF $\Leftrightarrow$ PDPF)**.** *We note that in regard to feasibility, this implication in fact goes in both directions. That is, existence of a privately puncturable PRF (P-PPRF) additionally implies the existence of a PDPF. Intuitively, a P-PPRF is precisely a PDPF but with* random*, versus chosen, payload. For small output domains (such as $\mathbb{Z}_2$), however, this can be addressed, e.g., by rejection sampling.*

Namely, given a P-PPRF, the corresponding $\mathsf{Gen}_0(1^\lambda, M, \hat{\mathbb{Z}}_2)$ will sample a random ("master") PRF key k_0. The algorithm $\mathsf{Gen}_1(k_0, \hat{f}_{\alpha,\beta})$ for a given point function $\hat{f}_{\alpha,\beta}$ will run independent executions of the randomized procedure $\mathsf{Punc}(k_0, M, \alpha)$ to generate a PRF key punctured at α, repeating until the resulting punctured key $k_1 \leftarrow \mathsf{Punc}(k_0, M, \alpha)$ yields the desired target offset $\mathsf{Eval}(k_0, M, \alpha) + \mathsf{PuncEval}(k_1, \alpha) = \beta$. The algorithms Eval_0 and Eval_1 of the PDPF then become the corresponding executions of Eval and $\mathsf{PuncEval}$ of the P-PPRF. Security follows from the privacy of the identity of the punctured input (intuitively, hiding α) together with pseudorandomness of the punctured evaluation on feasible output domain (intuitively, a punctured key for the real offset β is indistinguishable from a key for random $\beta' \leftarrow \mathbb{Z}_2$, since there are polynomially many possible offsets). And, since the output domain size is feasible, these algorithms remain polynomial time.

Overall, this close connection to P-PPRFs provides yet another motivation for the study of PDPFs.

4.2 DPF with Constant-Round Black-Box Distributed Gen

In this section we demonstrate that the techniques behind our PDPF construction can be used to give the first (standard) DPF construction (for feasible domain sizes) in which the key size is polylogarithmic in the domain size N, and whose key generation Gen admits a particularly efficient *secure distributed generation* procedure. Namely, the distributed Gen protocol makes only black-box use of OT and a PRG, and executes in a fixed *constant round complexity.* Concretely, we show that 5 rounds suffice.

As with the previous sections, the runtime of our DPF Eval algorithm (as well as $\mathsf{EvalAll}$) will be linear in the domain size N. Note that in this section, however, our DPF Gen procedure will only be logarithmic in N.

Concretely, by "distributed Gen," we refer to a secure computation protocol between two parties. We consider only security against a semi-honest adversary (i.e., who follows the protocol as prescribed but attempts to extrapolate information beyond its own input and output). The input consists of the desired security parameter 1^λ and input/output domain descriptions of the desired point function as common input, as well as *secret shares* of the desired point function values α and β over the respective spaces. The output is a randomly sampled key pair $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, \hat{f}_{\alpha,\beta})$, where each party learns its corresponding key.

Theorem 7 (Constant-round distributed Gen)**.** *There exists a small-domain DPF* $(\mathsf{Gen}, \mathsf{Eval})$*, with key size* $\mathrm{poly}(\lambda, \log N)$*, where* Gen *on secret-shared α, β can be implemented by a 5-round protocol making only a black-box use of oblivious transfer and a pseudorandom generator.*

The DPF is based on our PDPF construction from Corollary 2: Given a point function $\hat{f}_{\alpha,\beta}$, the DPF keys are formed via $\mathrm{poly}(\lambda, \log N)$ punctured PRFs, each serving as a ϵ-secure PDPF for some related $\hat{f}_{\alpha_i,\beta_i}$. The choice of the values (α_i, β_i) is computable via a small non-cryptographic randomized circuit as a function of α, β. For simplicity we present the results for fixed payload $\beta = 1$ and output space $\mathbb{Z}$; however, our construction extends naturally.

The main departure from our PDPF is that for each ϵ-secure DPF, instead of puncturing the corresponding PRF key k_i at a random input x_i^* *with the desired evaluation* $\mathsf{PRF.Eval}(k_i, x_i^*) = \alpha_i$, we will instead simply puncture the PRF at a completely random x_i^*, and provide both parties with the offset $\Delta_i = (\mathsf{PRF.Eval}(k_i, x_i^*) - \alpha_i)$. Recall that puncturing at x_i^* corresponds to an ϵ-secure DPF for $\alpha' = \mathsf{PRF.Eval}(k_i, x_i^*)$. Thus the parties will simply "shift" all evaluations by this offset Δ_i, effectively converting it to a DPF on α_i. This is possible due to the communication with *both* parties, which leads to computation being only logarithmic in N, as opposed to being linear in N in "1.5-server" regime, where we cannot afford online communication with both parties.

Consider the security of this modified scheme. Since the PPRF is now punctured at a random input, independent of any of its PRF evaluations, the punctured key (corresponding to DPF key k_1) now directly hides the punctured evaluation; thus, the offset Δ_i completely hides the secret value α_i. On the other hand, given the PRF key (corresponding to DPF key k_0), the evaluation of the PRF on a random input has a close-to-uniform, but biased distribution, corresponding to the unequal representation of different output values. This will yield the inverse-polynomial ϵ security for the corresponding DPF (where α_i is masked by a biased one-time pad). Here the bias ϵ is precisely as in the statistical balls and bins analysis from Lemma 1 in the PDPF analysis.

Note that this offset-to-random simplifies the key generation procedure (e.g., the cost of Gen no longer scales with the full domain size N), and adds only minor cost in regard to computation and key size. The reason this was not used in the prior sections is because the resulting construction is no longer a *programmable* DPF, which in particular requires the first key k_0 to be completely independent of the point function to be shared. However, this intermediate version is also a compelling construction offering alternative complexity tradeoffs.

Given this modified DPF construction, the new Gen procedure takes the following form. We mark by (*) those steps whose computation requires evaluation of a cryptographic PRG; all other computations are non-cryptographic.

$\mathsf{Gen}(1^\lambda, \hat{f}_\alpha)$, where $\alpha \in [N]$:

1. Compute the randomized mapping $(\alpha_1, \dots, \alpha_q) \leftarrow d(\alpha)$, where $d : [N] \to [L]^q$ is as in Lemma 2 (security amplification).[8]
2. Sample q random PPRF keys: $k_1, \dots, k_q \leftarrow \{0,1\}^\lambda$.
3. For each $i \in [q]$:
   (a) (*) Generate a punctured key $k_i^* \leftarrow \mathsf{Punc}(k_i, x_i^*)$, for random input x_i^*.
   (b) (*) Compute the punctured evaluation $\alpha_i' = \mathsf{PRF.Eval}(k_i, x_i^*)$.
   (c) Compute offset $\Delta_i = \alpha_i' - \alpha_i$
4. Output DPF keys $K_0 = ((k_1, \Delta_1), \dots, (k_q, \Delta_q))$ and $K_1 = ((k_1^*, \Delta_1), \dots, (k_q^*, \Delta_q))$.

Consider now a protocol Π_{Gen} for securely evaluating distributed Gen, where parties know only secret shares of α and must learn only their own resulting DPF key. Note that each non-cryptographic computation step can be securely evaluated in constant rounds and making only black-box use of oblivious transfer by using generic secure computation techniques.

This leaves two additional steps to address: puncturing the PPRF keys, and computing (secret shares of) the evaluations of the PRFs at the punctured inputs. Note that the latter can be done directly if one party holds the full PRF key k_i and the other party holds the punctured PRF key k_i^*, by each simply computing the sum of all computable PRF output values, which differ precisely by the punctured output. For the former step, of puncturing the PPRF keys, we observe that a two-round protocol for *precisely* this task were presented in the works of [10, 43] (within the context of an application of PPRFs to pseudorandom correlation generators for the OT correlation), applying the techniques of the Doerner-shelat protocol for DPFs [28] to the simpler setting of PPRFs. Intuitively, in order to puncture one PPRF key, the protocol consists of a collection of string OTs executed in parallel, one for each level in the evaluation tree of the PPRF, where the selection bits correspond to the bits of the punctured input x^*, and the message strings are computable as a function of the partial PRF evaluations at the given level. In particular, the protocol supports direct secure parallel composition of multiple instances.

Theorem 8 ([10, 43])**.** *Consider the GGM-based PPRF construction of [7, 36, 17]. There exists a two-round secure two-party protocol Π_{Punc} making only a black-box use of oblivious transfer and a pseudorandom generator, for evaluating the functionality with parties' inputs $((k_i)_{i\in[q]}, (x_i^*)_{i\in[q]})$ and outputs $(\perp, (k_i^*)_{i\in[q]})$, where each $k_i^* = \mathsf{Punc}(k_i, x_i^*)$.*

[8]Note that d is non-cryptographic. Concretely, for the case of Reed-Muller locally decodable codes, the mapping d corresponds to effectively generating Shamir secret shares of the input value α.

We next describe the constant-round distributed Gen protocol, making use of Π_{Punc} (and, in turn, the GGM-based PPRF). In the protocol description we refer to the two parties as P_0 and P_1.

Distributed Gen protocol, Π_{Gen}:
Inputs: Common: 1^λ, domain size N. P_0, P_1 hold secret shares α^0, α^1 of $\alpha \in [N]$.[9]

1. Party P_0 locally samples q random PPRF keys: $k_1, \ldots, k_q \leftarrow \{0,1\}^\lambda$.
2. Party P_1 locally samples q random PPRF inputs $x_1^*, \ldots, x_q^*$.
3. Parties P_0, P_1 jointly execute q parallel executions of protocol Π_{Punc}, on respective inputs $(k_i)_{i\in[q]}$ and $(x_i^*)_{i\in[q]}$. As output, party P_1 learns q punctured keys $(k_i^*)_{i\in[q]}$.
4. For each $i \in [q]$, each party locally computes the sum of all its computable PPRF evaluations: For P_0, this is $\sigma_i^0 = \sum_x \mathsf{PPRF.Eval}(k_i, x)$. For P_1, this is $\sigma_i^1 = \sum_{x \neq x_i^*} \mathsf{PPRF.PuncEval}(k_i^*, x)$, where sums are taken over $\mathbb{Z}_N$ (the domain space of the DPF).
5. The parties jointly perform a (generic) secure computation protocol for evaluating the following functionality:
   - Input: Each party P_b holds its original input share α^b and $(\sigma_i^b)_{i\in[q]}$.
   - Computation:
     (a) Evaluate the randomized mapping $(\alpha_1, \ldots, \alpha_q) \leftarrow d(\alpha^0 + \alpha^1) \in [L]^q$ from Lemma 2, where $\alpha^0 + \alpha^1$ represents the reconstructed value of the secret shared α (e.g., sum over $\mathbb{Z}_N$).
     (b) For each $i \in [q]$, compute $\Delta_i = (\sigma_i^0 - \sigma_i^1) - \alpha_i$. Recall σ_i^0 is equal to σ_i^1 plus the ith punctured evaluation.
   - Output: To both parties: $(\Delta_i)_{i\in[q]}$.

Security of the protocol Π_{Gen} follows by the security of the underlying Π_{Punc} and generic constant-round secure computation protocols. The round complexity of Π_{Gen} consists of (1) an execution of Π_{Punc}, in 2 rounds, followed by (2) the generic secure computation of a non-cryptographic functionality, in 3 rounds (note that both parties receive output). Thus, the combined round complexity is bounded by 5 rounds.

Comparison to Doerner-shelat[28]. As stated, the round complexity of our DPF distributed generation protocol is constant (5 rounds), as opposed to $\log N$ as in [28]. The communication complexity of our distributed Gen is also better than [28], due to the roughly $2\times$ improvement in our key size and an additive communication overhead in [28]. To give some data points, for $N = 10^5$, and $2^{-10} \leq \epsilon \leq 2^{-4}$ the communication complexity of a single data access in our scheme is in the range of 48-122 KB, while in [28] it is $\sim 240\,KB$.

The computational complexity of a data access is better than [28] for small values of N and large errors, but the situation is reversed as N grows and the linear scan of N data items in [28] vs. the M data items in our scheme dominates. In [28] the access time for $10^3 \leq N \leq 10^5$ is in the range 15-20 ms, while in our scheme the access time is lower for the pairs ($N = 10^3$, $\epsilon = 10^{-8}$), ($N = 20 \cdot 10^4$, $\epsilon = 2^{-6}$), and ($N = 10^5$, $\epsilon = 2^{-4}$), but is higher for each N when ϵ is lower than the quoted figure.

4.3 Compressing DPF Correlations

In this section we discuss an application of PDPFs for compressing correlated randomness in certain secure computation applications.

[9]This secret sharing can be over $\mathbb{Z}_N$, bitwise over $\mathbb{Z}_2$, or otherwise, with insignificant effect for the given protocol. We describe w.r.t. shares over $\mathbb{Z}_N$ for simplicity.

Standard DPFs have a variety of applications in the context of secure 2-party computation (2PC). For instance, they serve as crucial building blocks for concretely efficient 2PC of RAM programs [28] or for pseudorandom correlation generators (PCGs) of truth-table correlations [12] and (authenticated) multiplication triples [11].

As an example, suppose the two parties would like to securely evaluate a circuit which consists of arbitrary n-gates $g : \{0,1\}^n \to \{0,1\}$ (e.g., computing the AND or the majority of the n input bits). Using instances of a random OT correlation, the communication complexity of mapping a secret-shared input to a secret-shared output is linear in the circuit size of g and and the round complexity is linear in the circuit depth. But given a random DPF correlation, this only requires n communication bits per party and a single communication round [35, 26]. Concretely, a random DPF correlation consists of secret-sharing of a random $\alpha \in \mathbb{Z}_N$, for $N = 2^n$, and a pair of keys $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, \hat{f}_{\alpha,1})$ where $\hat{f}_{\alpha,1} : \mathbb{Z}_N \to \mathbb{Z}_2$. The idea is that the DPF correlation can be locally expanded into a truth-table correlation [12], which can in turn be used to evaluate a g-gate with minimal online communication and round complexity.

Given many independent instances of a DPF correlation, one can obtain a generic speedup for 2PC of Boolean circuits by grouping small sets of Boolean gates into bigger g-gates [25]. This strongly motivates the goal of generating many independent instances of a random DPF correlation with low communication cost. However, there are no known practical methods for achieving this.

We observe that PDPF can be used to solve this problem in the following "trusted-offline" setting for 2PC. In an offline phase, Alice owns a long-term secret s (say, a secret key for encryption, identification, or signature). To eliminate a single point of failure, she splits s into two shares, s_A and s_B, sending s_B to Bob and keeping s_A to herself. She then erases all information except s_A. In the online phase, the parties receive online inputs P_i (resp., ciphertexts to decrypt, nonces for identification, or messages to sign) and wish to securely compute $f(s, P_i)$ for $i = 1, 2, \ldots, t$.

The key observation is that Alice can be fully trusted in the offline phase, since if she is corrupted before erasing s then the long-term secret is entirely compromised. In fact, if P_i is public, then s is the only secret in the system. Consequently, we trust Alice to generate pairs of DPF keys (k_0^j, k_1^j) in the offline phase, offload the keys k_0^j to Bob, and keep k_1^i. However, the communication cost of generating DPF instances for evaluating many g-gates is high.

A PDPF can provide a dramatic efficiency improvement in this scenario. To generate T independent instances of a DPF correlation, Alice generates and communicates only a single reusable offline key k_0 to Bob (128 communication bits in practice). Then, for each j, she generates an online key k_1^j for a point function $f_{\alpha^j,1}$ using the PDPF algorithm Gen_1. She also derives Bob's (fresh) $\mathbb{Z}_N$-share of α^j from the offline key and computes its own share α_1^j. In the end of the silent generation process, Alice erases all information except her DPF correlation entries (k_1^j, α_i^j). Now the two parties hold T compressed instances of a truth-table correlation that can be silently expanded just when needed.

Viewed more abstractly, the above PDPF-based solution yields a PCG for generating T instances of a size-N truth-table correlation, where one of the keys is of size λ and the other is of size $\approx T \cdot \lambda \log N$. Thus, if Alice acts as a PCG dealer (who is only trusted during the offline phase), the communication cost is constant in T and N and the storage cost grows logarithmically with N. This should be contrasted with two alternative solutions: (1) using a standard DPF, both PCG keys are of size $\approx T \cdot \lambda \log N$, and so the communication cost is high when T is large; (2) using a naive PDPF, with online key linear in the domain size, keeps Bob's key (communication) small, but requires Alice's key (storage) to grow linearly with $T \cdot N$ instead of $T \cdot \log N$. A similar improvement is relevant to other applications of DPF in 2PC, including silent generation of multiplication triples [11] or low-communication simulation of RAM programs [28].

Concrete efficiency. We make a few remarks about the concrete efficiency of using PDPF to generate truth-table correlations. First, because the above applications only require *random* DPF instances (where α is chosen at random), the computational cost of the PDPF key generation is comparable to a standard DPF. Second, while the PDPF evaluation of our constructions is only concretely efficient for moderate values of N and ϵ (see Section 5), this can be good enough for applications. In particular, even a relatively high value of ϵ (say $\epsilon = 2^{-6}$) only amounts to a tiny (and easily quantifiable) leakage in the spirit of differential privacy, which is often considered tolerable. Functions with a small truth-table size N arise in many application scenarios, including S-box computations in distributed evaluation of block ciphers (cf. [26]) or nonlinear activation functions in low-precision Machine Learning algorithms (cf. [2]).

PDPF correlations vs. FSS correlations. The truth-table correlations we generate via PDPF are quite broadly applicable, since they effectively allow using a richer set of (small-domain) gates instead of just standard Boolean or arithmetic gates (see [25, 26]). Their main disadvantage is the computational overhead inherited from the evaluation algorithm of our PDPF, which scales linearly with the truth-table size N. This should be contrasted with the recent use of FSS correlations for secure computation with preprocessing [15, 8], in which the computation cost scales logarithmically with N. However, in applications where the value of N is moderate, this computation overhead may not form an efficiency bottleneck.

4.4 Big Payload Optimization

The PDPF in Figure 1 has binary output, which is insufficient for certain applications. In this section, we construct a PDPF with a *random* output in $\mathbb{Z}_{2^\ell}$ for some parameter ℓ, which easily extends to random output in $\mathbb{Z}_m$ for any $1 \leq m \leq 2^\ell$. Applications of PDPF that work in an offline-online mode can either directly use random output values or correct a random value to a specified value at very low cost in the online stage, see Section 4.

Compared to the scheme for binary output this new construction increases the key size by roughly $O(\ell)$ and server computation time by roughly $O(\ell^2)$, improving over the natural alternatives that have either a much longer $N\ell$-bit key or a similar-sized key but server computation time that is $O(\ell^3)$ greater than the time required in the binary case.

The first of the natural alternatives that we mentioned is to provide the offline party with the entire truth-table of the function, and to give the online party the punctured truth-table. The key size in this case is $N\ell$. The second alternative is to repeat the scheme of Figure 1 ℓ times independently and then locally convert the secret-shared bits to secret-shared elements in $\mathbb{Z}_{2^\ell}$. The downside of this scheme is that to achieve error probability ϵ it is necessary to increase the GGM tree size and thus the overall evaluation time by a factor of $O(\ell^3)$ compared to the tree size of the scheme with binary output and the same error probability.

In our optimized PDPF scheme with large, random output, each output of the PRF is a pair of pseudorandom values (x, y) such that $x \in [N]$ and $y \in \mathbb{Z}_{2^\ell}$. The offline party again receives a short key that describes the PRF, while the online party receives a key that describes the PRF with t punctured points (α, y). Consider the value $\sum_{j=1}^{T} y_{i_j} \bmod 2^\ell$, where the PRF (or PPRF) evaluation of the whole input domain resulted in T values of the form (x, y_{i_j}). The difference of these values between the two parties is $0 \bmod 2^\ell$ for all $x \neq \alpha$ and is equal to the sum of y_{i_j} in the t punctured points for $x = \alpha$.

To give some intuition on the security of the construction we first note that if the point function is $f_{\alpha,\beta}$ then the offline party should have no information on α, since it does not know which points are punctured, and the online party should have no information on β due to the security of punctured PRFs. Hiding α from the online party is achieved by exploiting the difference between the expected number of leaves with first value α in the full tree, which we denote by T, and the number of punctured points which we denote by t. For a large enough T, the probability of distinguishing between the number of leaves with value α and the number of with value α' decreases below a required security threshold. Statistically hiding β from the offline party requires both that there is sufficient entropy in the random choice of t punctured points out of T possible points to ensure that the sum of y_{i_j} is close to a uniformly random element in $\mathbb{Z}_{2^\ell}$.

The scheme uses a Multi-Punctured PRF at t points, which is a natural generalization of PPRF.

Definition 7. *We say that a PPRF is a* t-puncturable PRF *if there exist additional polynomial time algorithms* $\mathsf{MPunc}(k, M, N, x_1, \ldots, x_t)$ *and* $\mathsf{MPuncEval}(k_p, x_1, \ldots, x_t)$*, which generalize* Punc *and* $\mathsf{PuncEval}$ *in the natural way.* $(\mathsf{Eval}, \mathsf{Punc}, \mathsf{PuncEval})$ *should satisfy the following additional requirements:*

- ***Correctness:*** *For every $\lambda, M, N \in \mathbb{N}$ and $x \neq x_1, \ldots, x_t \in [M]$ it holds that*

$$\Pr\left[\begin{matrix} k \xleftarrow{\$} \{0,1\}^\lambda, \\ k_p \leftarrow \mathit{MPunc}(k, M, N, x_1, \ldots, x_t) \end{matrix} : \mathit{MPuncEval}(k_p, x) = \mathsf{Eval}(k, M, N, x)\right] = 1.$$

- ***Security:*** *The adversary gives challenge domain sizes and input $(M, N, x_1, \ldots, x_t)$, the challenger draws $b \xleftarrow{\$} \{0,1\}$ and computes*

$$k \xleftarrow{\$} \{0,1\}^\lambda, \quad k_p \leftarrow \mathit{MPunc}(k, M, N, x_1, \ldots, x_t),$$
$$(y_{0,1}, \ldots, y_{0,t}) \xleftarrow{\$} [N]^t, \quad (y_{1,1}, \ldots, y_{1,t}) \leftarrow \mathsf{Eval}(k, M, N, x_1, \ldots, x_t).$$

- *The adversary outputs a guess $b' \leftarrow \mathcal{A}(k_p, y_b)$.*

For advantage bound $\epsilon(\lambda)$ and polynomial circuit size bound $S(\lambda)$, we say that the PPRF (Eval, *Punc*, *PuncEval*) *is $\epsilon(\lambda)$-secure if for any non-uniform adversary $\mathcal{A}$ of size $S(\lambda)$, it holds in the above experiment that* $\Pr[b = b'] - 1/N \leq \epsilon(\lambda)$.

The following proposition which generalizes Theorem 3 is implicit in the literature.

Proposition 6. *Given black-box access to a length-doubling PRG with seed length λ, there exists a secure t-puncturable PRF scheme, with input domain $[M]$ and output domain $[N]$, $N \leq 2^\lambda$, which can be implemented with punctured key length $|k_p| = \lambda t \log \frac{M}{t}$.*

Proof. The PRF is generated from a seed to the PRG via the GGM tree construction. The punctured key k_1 includes every sibling of a node on one of the t paths. The number of these nodes is $t \log \frac{M}{t}$, see for example the analysis in [39] Theorem 2. □

Theorem 9 (Big-payload optimization)**.** *Assume either an ideal PPRF or black-box access to a length-doubling PRG with λ-bit seed in the CRS model. There exists an ϵ-secure PDPF for sharing $f_{\alpha,\beta} : [N] \to \mathbb{Z}_{2^\ell}$ for a specified α and a random β, with the following efficiency costs:*

- *Online key size $|k_1| = O\left(\lambda t \log \frac{tN}{\epsilon^2}\right)$, for $t = \ell + 2 \log \frac{1}{\epsilon}$.*
- Gen *makes $O(tN \log \frac{t}{\epsilon^2})$ PRG invocations on average.*
- EvalAll *makes $O(\frac{Nt^2}{\epsilon^2})$ PRG invocations in the worst case.*

Proof. Consider the PDPF in Figure 3. Its correctness is immediate, and to prove its ϵ-security we separately analyze the security for the two parties.

The offline party has no information on α. To bound the information leakage on β, note that $\beta \equiv \sum_{j=1}^{t} y_{i_t} \bmod 2^\ell$, where $i_1, \ldots, i_t$ are chosen at random out of all the points $i_1, \ldots, i_{T_\alpha} \in [M]$ on which the PRF evaluates to $(\alpha, y), y \in \mathbb{Z}_{2^\ell}$. Even given α, the offline party doesn't have information on which t points were punctured out of the T_α possible points.

For the offline party this is exactly a subset-sum problem. [34] proves that the mapping from a subset of random values to its sum defines a family of universal hash functions on $\mathbb{Z}_{2^\ell}$, and therefore the Leftover Hash Lemma [33] applies. In other words, if the min-entropy of the random variable that chooses the subsets is at least $\ell + 2log\frac{1}{\epsilon}$ then there is at most ϵ probability to distinguish β from a random element in $\mathbb{Z}_{2^\ell}$.

In the ideal PRF model the values $y_1, \ldots, y_{T_\alpha}$ are sampled uniformly and independently in $\mathbb{Z}_{2^\ell}$ and assuming the existence of a PRG, these values are computationally indistinguishable from uniformly random elements. Therefore, the min-entropy of the random variable that models the selection of the punctured points is $\log \binom{T_\alpha}{t}$. The average value of T_α is $M/N \geq t^2$. It follows from the choice of t, that $log\binom{T_\alpha}{t} \geq \log t^t = t \log t \geq \ell + 2log\frac{1}{\epsilon}$, which proves the security for the offline party.

```
$\mathsf{Gen}_1(k_0 = (k_*, N, \ell, \hat{\mathbb{G}} = \mathbb{Z}_{2^\ell}), \alpha\epsilon)$:
Let $G : \{0,1\}^\lambda \to \{0,1\}^{\max\{2\lambda, \lambda+\ell\}}$ be a PRG.
    • Find minimal $t$ such that $t \log t \geq \ell + 2 \log \frac{1}{\epsilon}$
    • Compute $M = Nt^2/\epsilon^2$
    • Let
          $I \leftarrow \{m \in [M] : \mathsf{PPRF.Eval}(k_*, M, N+1, m) = (\alpha, y),\ y \in \mathbb{Z}_{2^\ell}\}$
    • Choose random $i_1, \ldots, i_t \in I$
    • Let $k_p \leftarrow \mathsf{PPRF.MPunc}(k_*, M, N+1, i_1, \ldots, i_t)$
    • output $k_1 = k_p$.

$\mathsf{EvalAll}_0(k_0 = (k_*, N, \ell, \hat{\mathbb{G}} = \mathbb{Z}_{2^\ell}))$:
    • For every $x \in [N]$, simultaneously compute
        – $S_x \leftarrow \{m \in [M] : \mathsf{PPRF.Eval}(k_*, M, N+1, m) = (x, y),\ y \in \mathbb{Z}_{2^\ell}\}$
        – Compute $H_{x,0} = \sum_{(x,y) \in S_x} y \bmod 2^\ell$
    • Output $H_0 = (H_{x,0})_{x \in [N]}$.

$\mathsf{EvalAll}_1(k_1)$:
    • For every $x \in [N]$, simultaneously compute
        – $S_x \leftarrow \{m \in [M] : \mathsf{PPRF.MPuncEval}(k_1) = (x, y),\ y \in \mathbb{Z}_{2^\ell}\}$
        – Compute $H_{x,1} = -\sum_{(x,y) \in S_x} y \bmod 2^\ell$
    • Output $H_1 = (H_{x,1})_{x \in [N]}$.
```

Figure 3: Small-domain and large output computationally $1/\text{poly}(\lambda, N)$-secure PDPF for point functions with output $\beta \in \mathbb{Z}_{2^\ell}$ and domain size N. The leaves of the PRF are pairs $(x, y) \in [N] \times \mathbb{Z}_{2^\ell}$.

The online party does not get information on β, either unconditionally in the ideal PRF model or computationally, assuming the existence of a PRG. To bound the information it gets on α, consider the bound of Lemma 6 on a balls and bins experiment, which exactly reflects the security game for the online party. The experiment is requires to determine which of two designated values α and α' are punctured in the PPRF, given only their relative frequencies in the points that weren't punctured.

The expected sum of the number of α and α' "balls" in the punctured key is $2T-t$ for $T = M/N = t^2/\epsilon^2$. The event that the actual number of balls is too far from the expected value has negligible probability and we discount it in the rest of the analysis. Concretely, we assume that the actual number of balls is within $(2-4/\pi)t^2/\epsilon^2 - t$ of the expected t^2/ϵ^2. Therefore, if the actual number of balls is w then $w \geq \frac{4t^2}{\pi\epsilon^2}$ and if $A(w) \sim Binomial(w; 1/2)$ then by a well-known approximation $\Pr[A(w) = w] \approx \frac{1}{\sqrt{\pi w}}$ and therefore,

$$\Pr\left[A(w) \in \left\{\frac{w-t}{2}, \frac{w-t+1}{2}, \ldots, \frac{w+t-1}{2}\right\}\right] \leq \frac{2t}{\sqrt{\pi w}} \leq \epsilon.$$

It follows that the statistical distance between the two distributions of α or α' being the punctured point, is at most ϵ.

The length of the punctured key k_1 is $\lambda t \log \frac{M}{t}$ by Proposition 6, which is $\lambda t \log \frac{tN}{\epsilon^2}$. EvalAll constructs the whole tree and therefore the number of PRG invocations it makes is $2M = \frac{Nt^2}{\epsilon^2}$. Gen can use the same probabilistic "lazy" approach used in the binary case of evaluating the PRF on random input points in $[M]$ until t of them have first coordinate α. Since the probability that each point m evaluates to α is $1/N$ (up to a negligible difference in the CRS model), the expected number of points that needs to be evaluated is tN. The expected number of PRG invocations in Gen is thus the sum of the nodes along the paths to tN, which is identical to the length of a key for tN punctured points (divided by the node description length, λ). Therefore, by Proposition 6 the expected number of PRG invocations in Gen is $O(tN \log \frac{M}{tN} = tn \log \frac{t}{\epsilon^2})$.

$\square$

Remark 3. *Theorem 9 highlights the asymptotic advantage in key size of the optimized construction compared to the trivial PDPF, in which key size is $|k_1| = N\ell$. However, in concrete terms that advantage becomes apparent as N grows and with current techniques the full evaluation time is relatively high, even though it is an improvement over repeating the binary PDPF ℓ times. To give two data points on the performance of our construction, for $N = 20000, \epsilon = 2^{-6}$ and $\ell = 10$, the key length is $|k_1| = 10$ KB,* Gen *time is* 3.8 *ms, but* EvalAll *time is* 13.3 *sec. In the same setting but with $\ell = 20$ the key size is* 14.8 *KB,* Gen *time is* 8.1 *and* EvalAll *time is* 39.7 *sec.*

The PDPF scheme in Figure 3 has the disadvantage that it outputs a sharing of a point function $f_{\alpha,\beta}$ for a specified α, but a random β. We next show how to convert that that scheme to a DPF scheme in which both α and β are deterministic. Note that this is a DPF rather than a PDPF scheme, i.e. both α and β are specified during Gen.

The idea is to provide both parties with an additional *correction word* CW that is a function of β and the punctured values $y_{i_1}, \ldots, y_{i_t}$. CW will be applied by each party with opposite signs in the evaluation of $x \neq \alpha$, but will exactly shift the evaluation of the DPF at α to share β.

In more detail, first let t be odd, so that it has an inverse modulo 2^ℓ (t can always be increased). Then, set $CW = t^{-1}[\beta - \sum_{j=1}^{t} y_{i_j}] \bmod 2^\ell$ and provide CW to both parties together with the keys.

Let all the PRF values with first coordinate x be $(x, y_1), \ldots, (x, y_{T_\alpha})$ for every $x \in [N]$. When evaluating x, both parties compute $W_x = \sum_{i=1}^{T_x} y_i + T_x \cdot CW \bmod 2^\ell$. The offline party outputs W_x and the online party outputs $-W_x \bmod 2^\ell$. This ensures that the difference between the output values is 0 for $x \neq \alpha$ and is β for $x = \alpha$.

5 Concrete Efficiency

In this section we compare the concrete efficiency of our construction from Theorem 4 to a naive PDPF construction. For our comparison we will consider PDPFs over $\mathbb{G} = \mathbb{Z}$ and $\beta \in \mathbb{G}' = \{0, 1\}$. Throughout the section we model the PPRF as an ideal PPRF, see Appendix C for further analysis.

While Theorem 4 gives a $\epsilon \approx \sqrt{N/M}$ security bound, we empirically find that the real statistical distance in the statistical variant of the balls-and-bins experiment, as in Lemma 1 is $\epsilon \approx 0.564\sqrt{N/M}$, and we use this estimate in the tables below. For estimating the running time of $\mathsf{EvalAll}$ in our construction we use Theorem 3, by which $\mathsf{EvalAll}$ makes $(M \log_2(N+1)/\lambda$ PRG calls. In addition, by Proposition 4, Gen_1 makes $((N+1)\log_2 M)/\lambda$ PRG calls.

The naive PDPF construction is obtained by having $\mathsf{EvalAll}_0$ treat k_* (obtained by running Gen_0) as a PRF key, expanding it to a truth table of length N over $\{0, \ldots, \lceil 1/\epsilon \rceil - 1\}$ for an integer $1/\epsilon$. Denote by $f^0 : [N] \to \mathbb{Z}$ the function with this truth table. Then, Gen_1 will generate k_1 by simply computing the truth table of the function $f^1 = f_{\alpha,\beta} - f^0$ (hence $|k_1| = N\lceil \log_2(1/\epsilon) \rceil$), and $\mathsf{EvalAll}_1$ will output the truth table it got. Note that this naive PDPF construction is ϵ-secure. Because both Gen and $\mathsf{EvalAll}$ compute the PRF on all points, by Theorem 3, they make $(2N-1)(\log N)/(2\lambda)$ PRG calls.

Remark 4 (Privacy and key length for the naive PDPF)**.** *The naive construction provides* negligible *privacy error and online key of $N \cdot \log |\mathbb{G}|$ for output group $\mathbb{G}$. In aggregation-type applications, one either needs to pick a very large finite $\mathbb{G}$ or use the group of integers $\mathbb{Z}$ with key size $N \cdot c$ and settle for 2^{-c}-privacy. To make the comparison meaningful, we went for the latter option with $\epsilon = 2^{-c}$.*

In Table 1 we compare the key size and running time of Gen and $\mathsf{EvalAll}$ of our PDPF to the naive PDPF, for fixed $\lambda = 128$. Our time unit is PRG evaluations, assuming $1.8 \cdot 10^8$ evaluations per second of $G : \{0,1\}^{128} \to \{0,1\}^{256}$.

Acknowledgements

We thank the CRYPTO reviewers for many useful comments and suggestions, including a simplification of the proof of Theorem 4. We thank Katharina Boudgoust and Mark Simkin for their helpful comments. Elette Boyle was supported by AFOSR Award FA9550-21-1-0046, ERC Project HSS (852952), ERC Project NTSC (742754), and a Google Research Scholar Award. Niv Gilboa was supported by ISF grant 2951/20, ERC grant 876110, and a grant by the BGU Cyber Center. Yuval Ishai was supported by ERC Project NTSC (742754), BSF grant 2018393, and ISF grant 2774/20. Victor I. Kolobov was supported by ERC Project NTSC (742754) and ISF grant 2774/20.

References

[1] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder - scalable, robust anonymous committed broadcast. In *CCS '20*, pages 1233–1252, 2020.

[2] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. QUOTIENT: two-party secure neural network training and prediction. In *ACM CCS 2019*, pages 1231–1247, 2019.

[3] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, page 850–864, November 1984.

[4] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. pages 762–776, 2021.

[5] Dan Boneh, Sam Kim, and Hart William Montgomery. Private puncturable prfs from standard lattice assumptions. In *EUROCRYPT 2017*, pages 415–445, 2017.

[6] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In *PKC 2017*, 2017.

[7] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013*, pages 280–300, 2013.

ϵ/N	1000	20000	100000
2^{-4}	$\mathbf{0.3\,KB}/0.5\,KB$ $1.5\,\mu s, \mathbf{0.002\,\mu s}/0.4\,\mu s$ $38.8\,\mu s/0.4\,\mu s$	$\mathbf{0.3\,KB}/9.8\,KB$ $42.1\,\mu s, \mathbf{0.005\,\mu s}/12.4\,\mu s$ $1.1\,ms/12.4\,\mu s$	$\mathbf{0.4\,KB}/48.8\,KB$ $242.6\,\mu s, \mathbf{0.006\,\mu s}/72.1\,\mu s$ $6.2\,ms/72.1\,\mu s$
2^{-6}	$\mathbf{0.3\,KB}/0.7\,KB$ $2.5\,\mu s, \mathbf{0.002\,\mu s}/0.4\,\mu s$ $621.2\,\mu s/0.4\,\mu s$	$\mathbf{0.4\,KB}/14.7\,KB$ $68.7\,\mu s, \mathbf{0.005\,\mu s}/12.4\,\mu s$ $17.3\,ms/12.4\,\mu s$	$\mathbf{0.4\,KB}/73.2\,KB$ $395.5\,\mu s, \mathbf{0.006\,\mu s}/72.1\,\mu s$ $99.7\,ms/72.1\,\mu s$
2^{-8}	$\mathbf{0.4\,KB}/1.0\,KB$ $3.4\,\mu s, \mathbf{0.002\,\mu s}/0.4\,\mu s$ $9.9\,ms/0.4\,\mu s$	$\mathbf{0.5\,KB}/19.5\,KB$ $95.2\,\mu s, \mathbf{0.005\,\mu s}/12.4\,\mu s$ $276.8\,ms/12.4\,\mu s$	$\mathbf{0.5\,KB}/97.7\,KB$ $548.3\,\mu s, \mathbf{0.006\,\mu s}/72.1\,\mu s$ $1.6\,sec/72.1\,\mu s$
2^{-10}	$\mathbf{0.4\,KB}/1.2\,KB$ $4.4\,\mu s, \mathbf{0.002\,\mu s}/0.4\,\mu s$ $159.0\,ms/0.4\,\mu s$	$\mathbf{0.5\,KB}/24.4\,KB$ $121.8\,\mu s, \mathbf{0.005\,\mu s}/12.4\,\mu s$ $4.4\,sec/12.4\,\mu s$	$\mathbf{0.6\,KB}/122.1\,KB$ $701.2\,\mu s, \mathbf{0.006\,\mu s}/72.1\,\mu s$ $25.5\,sec/72.1\,\mu s$

Table 1: Key size, running time of Gen_1 and $\mathsf{EvalAll}$ of our PDPF construction from Theorem 4 (left) compared to the naive one (right). For the PDPF from Theorem 4 there are two Gen_1 running times, the smaller one corresponding to time needed to generate a key for a *random point function*. Running times are based on an AES-based PRG implementation benchmarked at $1.8 \cdot 10^8$ PRG calls per second on a single core. For $M = 0.318 \cdot N/\epsilon^2$ and $\lambda = 128$, in our construction, the key size is $\lambda \log_2 M$, $\mathsf{EvalAll}$ makes $(2M-1)(\log N)/(2\lambda)$ calls to the PRG, and Gen_1 makes $(N+1)(\log(M/N))(\log N)/(2\lambda)$ calls to the PRG (and $(\log(N))^2/(2\lambda)$ PRG calls for the random point function). In the naive construction, the key size is $N \log_2(1/\epsilon)$, and $\mathsf{EvalAll}$ and Gen_1 both make $(2N-1)(\log N)/(2\lambda)$ calls to the PRG.

[8] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *EUROCRYPT 2021, Part II*, pages 871–900, 2021.

[9] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector ole. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 896–912, 2018.

[10] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS*. ACM, 2019.

[11] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-lpn. In *CRYPTO 2020, Part II*, pages 387–416.

[12] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019*, 2019.

[13] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-lpn. In *Annual International Cryptology Conference*, pages 387–416. Springer, 2020.

[14] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *CCS*, 2016.

[15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography Conference*, pages 341–371, 2019.

[16] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In *CRYPTO 2022*, 2022.

[17] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC 2014*, pages 501–519.

[18] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, pages 662–693.

[19] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained prfs (and more) from LWE. In *TCC 2017, Part I*, pages 264–302.

[20] Ran Canetti and Yilei Chen. Constraint-hiding constrained prfs for nc^1 from LWE. In *EUROCRYPT 2017, Part I*, pages 446–476, 2017.

[21] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50. IEEE, 1995.

[22] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX symposium on networked systems design and implementation (NSDI 17)*, pages 259–282, 2017.

[23] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.

[24] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020,Part I*, 2020.

[25] Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In *EUROCRYPT 2019*, pages 473–503, 2019.

[26] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *CRYPTO 2017*.

[27] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 122–138. Springer, 2000.

[28] Jack Doerner and Abhi Shelat. Scaling oram for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 523–535, 2017.

[29] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *EUROCRYPT 2014*, pages 423–440, 2014.

[30] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT*, 2014.

[31] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, August 1986.

[32] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath T. V. Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with popcorn. In *USENIX*, 2016.

[33] Russell Impagliazzo, Leonid A Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 12–24, 1989.

[34] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of cryptology*, 9(4):199–216, 1996.

[35] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In *TCC 2013*, 2013.

[36] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *CCS 2013*, pages 669–684.

[37] Ueli M. Maurer, Krzysztof Pietrzak, and Renato Renner. Indistinguishability amplification. In *CRYPTO*, pages 130–149, 2007.

[38] Ueli M. Maurer and Stefano Tessaro. A hardcore lemma for computational indistinguishability: Security amplification for arbitrarily weak prgs with optimal stretch. In *TCC 2010*, pages 237–254, 2010.

[39] Dalit Naor, Moni Naor, and Jeff Lotspiech. Revocation and tracing schemes for stateless receivers. In *Crypto*, pages 41–62. Springer, 2001.

[40] Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. Spectrum: High-bandwidth anonymous broadcast with malicious security. *IACR Cryptol. ePrint Arch.*, page 325, 2021.

[41] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *STOC 1997*, pages 294–303, 1997.

[42] Chris Peikert and Sina Shiehian. Privately constraining and programming prfs, the LWE way. In *PKC 2018, Part II*, pages 675–701, 2018.

[43] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-ole: Improved constructions and implementation. In *ACM CCS*, pages 1055–1072, 2019.

[44] Elaine Shi, Waqar Aqeel, Balakrishnan Chandrasekaran, and Bruce M. Maggs. Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In *CRYPTO*, pages 641–669, 2021.

[45] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS 1982*, pages 80–91.

Supplementary Material

A Reusable DPF

Definition 8. *We say that a* $(2,1)$*-DPF* $(\mathsf{Gen}, \mathsf{Eval}_0, \mathsf{Eval}_1)$ *is a* reusable DPF*, or RDPF for short, if* Gen *can be decomposed into a pair of algorithms* $\Pi = (\mathsf{Gen}_0, \mathsf{Gen}_1)$ *with the following syntax:*

- $\mathsf{Gen}_0(1^\lambda, N, \hat{\mathbb{G}}) \to k_0$*: On input security parameter* λ*, domain size* N *and output group description* $\hat{\mathbb{G}}$*, returns a key* $k_0 = (k_*, N, \hat{\mathbb{G}})$*, where* $k_* \in \{0,1\}^\lambda$*.*

- $\mathsf{Gen}_1(k_0, \hat{f}_{\alpha,\beta}, L, \ell) \to k_1$*: On input key* $k_0 = (k_*, N, \hat{\mathbb{G}})$*, point function description* $\hat{f}_{\alpha,\beta} = (N, \hat{\mathbb{G}}, \alpha, \beta)$*, ID space size* L*, and ID* $\ell \in [L]$*, returns a key* $k_1^\ell \in \{0,1\}^*$*.*

Moreover, we require that k_**, returned by* Gen_0 *as part of* k_0*, is a uniform random string, namely,* $k_* \xleftarrow{\$} \{0,1\}^\lambda$*. The algorithm* Gen_1 *should satisfy the following (multi-message) security requirement instead of the one in Definition 2:*

- ***Security:*** *Consider the following semantic security challenge experiment:*

  1. *The adversary gives ID space size* $L \leftarrow \mathcal{A}(1^\lambda)$*, followed by challenge point function descriptions* $(\hat{f}^{0,\ell} = (N_0^\ell, \hat{\mathbb{G}}_0^\ell, \alpha_0^\ell, \beta_0^\ell), \hat{f}^{1,\ell} = (N_1^\ell, \hat{\mathbb{G}}_1^\ell, \alpha_1^\ell, \beta_1^\ell)) \leftarrow \mathcal{A}(1^\lambda)$ *with* $N_0^\ell = N_1^\ell$ *and* $\hat{\mathbb{G}}_0^\ell = \hat{\mathbb{G}}_1^\ell$*, for every* $\ell \in [L]$*.*
  2. *The challenger samples* $b \xleftarrow{\$} \{0,1\}, k_* \xleftarrow{\$} \{0,1\}^\lambda$ *and also, for every* $\ell \in [L]$*, assigns* $k_1^\ell \leftarrow \mathsf{Gen}_1(k_*^\ell, \hat{f}^{b,\ell}, \ell)$*.*
  3. *The adversary outputs a guess* $b' \leftarrow \mathcal{A}((k_1^\ell)_{\ell \in [L]})$*.*

  Denote by $\mathrm{Adv}(1^\lambda, \mathcal{A}) = \Pr[b = b'] - 1/2$ *the advantage of* $\mathcal{A}$ *in guessing* b *in the above experiment. The notions of security are defined similarly to Definition 2 with respect to* $\mathrm{Adv}(1^\lambda, \mathcal{A})$*.*

Lemma 5. *A PDPF exists if and only if an RDPF exists.*

Proof. It is not difficult to see that an RDPF implies a PDPF by choosing $\ell = L = 1$.

Now, let $\mathsf{PDPF} = (\mathsf{PDPF.Gen}_0, \mathsf{PDPF.Gen}_1, \mathsf{PDPF.Eval}_0, \mathsf{PDPF.Eval}_1)$ be a PDPF. To construct an RDPF, RDPF, let Gen_0, and Eval_1 be identical to that of PDPF. The existence of a PDPF implies OWFs [30], and consequently also the existence of a PRF, $\mathsf{PRF.Eval}$. $\mathsf{RDPF.Gen}_1$ on input $k_0 = (k_*, N, \hat{G}), \hat{f}_{\alpha,\beta}, L, \ell$ will compute $k_*^\ell = \mathsf{PRF.Eval}(k_*, 2^\lambda, L, \ell)$ and output $\mathsf{PDPF.Gen}_1((k_*^\ell, N, \hat{G}), \hat{f}_{\alpha,\beta})$. $\mathsf{RDPF.Eval}_0$ on input $k_0 = (k_*, N, \hat{G}), x$, will first compute $k_*^\ell = \mathsf{PRF.Eval}(k_*, 2^\lambda, L, \ell)$ and then output $\mathsf{PDPF.Eval}_0((k_*^\ell, N, \hat{G}), x)$.

The DPF correctness of PDPF implies the correctness of RDPF, because $\mathsf{RDPF.Eval}_0$ and $\mathsf{RDPF.Eval}_1$ essentialy compute $\mathsf{PDPF.Eval}_0$ and $\mathsf{PDPF.Eval}_1$ applied to the keys $(k_*^\ell, N, \hat{G})$ and $\mathsf{PDPF.Gen}_1((k_*^\ell, N, \hat{G}), \hat{f}_{\alpha,\beta})$, respectively, which are valid keys of PDPF.

Security follows because the truth table of the PRF is indistinguishable from random to polynomial sized adversaries, and so in the hybrid world all k_*^ℓ are independently and uniformly random. In this hybrid world the security game is played with respect to L independent instances of PDPF, which is secure if PDPF is secure. □

B Balls-and-bins experiment analysis

The following lemma generalizes the equality part of Lemma 1 to multiple balls removed from a single bin.

Lemma 6. *For integers* $M > N > 0$*, an integer* t*, and* $i, j \in [N]$*, let* D_i *and* D_j *be distributions over* $\{1, \ldots, N, \perp\}^M \cup \{\mathit{fail}_i, \mathit{fail}_j\}$ *of the locations of* M *balls independently and randomly thrown into* N *bins, such that we then change the position of a* t *balls, chosen randomly from bin* i *and bin* j*, respectively, to* $\perp$

(this corresponds to the ball's "removal" from the bin). If there is no ball in bin i, then the output of D_i is fail_i. *Then*

$$d(D_i, D_j) = \sum_{w=0}^{M} \binom{M}{w} \left(1 - \frac{2}{N}\right)^{M-w} \left(\frac{2}{N}\right)^{w} \Pr\left[A(w) \in \left\{\frac{w-t}{2}, \frac{w-t+1}{2}, \ldots, \frac{w+t-1}{2}\right\}\right]$$

where $A(w) \sim Binomial(w; 1/2)$.

Proof. For a configuration of balls into bins $C \in \{1, \ldots, N, \perp\}^M$ denote by $\#bin_i(C)$ the number of balls in bin i in this configratuion. Define the event $E = \{C \in \{1, \ldots, N, \perp\}^M : \#bin_i(C) \leq \#bin_j(C)\} \cup \{\mathsf{fail}_i\}$. We first argue that

$$\Pr_{D_i}[E] - \Pr_{D_j}[E] = d(D_i, D_j).$$

Indeed, define the event $E' = \{C \in \{1, \ldots, N, \perp\}^M : \Pr_{D_i}[C] \geq \Pr_{D_j}[C]\} \cup \{\mathsf{fail}_i\}$. It is not difficult to see that $\Pr_{D_i}[E'] - \Pr_{D_j}[E'] = d(D_i, D_j)$. Hence, it is sufficient to prove that $E = E'$. For a configuration $C \neq \mathsf{fail}_i, \mathsf{fail}_j$ we have that

$$\Pr_{D_i}[C] = \frac{1}{N^M \binom{\#bin_i(C)+t}{t}}$$

because we first need to throw M balls into N bins, and then the probability a specific t-sized set of balls is chosen is $1/\binom{\#bin_i(C)+t}{t}$. The same equality holds with respect to $\Pr_{D_j}[C]$. Thus,

$$\Pr_{D_i}[C] \geq \Pr_{D_j}[C] \iff \#bin_i(C) \leq \#bin_j(C),$$

and so $E = E'$. Hence, it is sufficient to calculate $\Pr_{D_i}[E] - \Pr_{D_j}[E]$ to estimate $d(D_i, D_j)$. We achieve this by coupling. To this end, let $(X, Y, Z) \sim Trinomial(M; 1/N, 1/N; 1 - 2/N)$. Then $\Pr_{D_i}[E] = \Pr[X - t \leq Y \vee X \leq t-1] = \Pr[X - t \leq Y]$ (second equality holds because $X \leq t-1 \Rightarrow X - t \leq Y$) and $\Pr_{D_j}[E] = \Pr[X \leq Y - t]$. Therefore,

$$\begin{aligned}
d(D_i, D_j) &= \Pr[Y - X \in \{-t, \ldots, t-1\} \\
&= \sum_{w=0}^{M} \Pr[Z = M - w] \Pr[Y - X \in \{-t, \ldots, t-1\} | X + Y = w] \\
&= \sum_{w=0}^{M} \binom{M}{w} \left(1 - \frac{2}{N}\right)^{M-w} \left(\frac{2}{N}\right)^{w} \Pr\left[A(w) \in \left\{\frac{w-t}{2}, \ldots \frac{w+t-1}{2}\right\}\right]
\end{aligned}$$

□

C Realizing Ideal PPRF

In this section we prove that a modification of the PDPF from Figure 1 in the CRS model is as secure (up to a negligible term) as the scheme where the PPRF is replaced by an *ideal PPRF*, which is a functionality that provides the online party with the result of a balls-and-bins experiment, where one ball is removed from a bin at random, and the bin is chosen according to the same strategy as Gen_1 in Figure 1.

In other words, for a given α, an algorithm that chooses the point to puncture Choose, and algorithm that evaluates the PRF on all points $\mathsf{EvalAll}$, and an algorithm which punctures the key Punc, the distribution of the real PDPF key together with the CRS, denoted by Real_α, is as follows:

1. $k \leftarrow \{0,1\}^\lambda$
2. $\mathsf{CRS} \leftarrow \mathbb{Z}_N^M$
3. $x \leftarrow \mathsf{Choose}(\mathsf{CRS} + \mathsf{EvalAll}(k), \alpha)$

$\mathsf{Gen}_1(k_0 = (k_*, N, \hat{\mathbb{G}}, \hat{\mathbb{G}}'), \hat{f}_{\alpha,\beta} = (N, \hat{\mathbb{G}}, \alpha, \beta), \mathsf{CRS} \in \mathbb{Z}_{N+1}^M)$:

- If $\beta = 1$ then find all indices
$$L \leftarrow \{\ell \in [M] : \mathsf{PPRF.Eval}(k_*, M, N+1, \ell) + \mathsf{CRS}_\ell = \alpha\}\,.$$
- Else, if $\beta = 0$ then find all indices
$$L \leftarrow \{\ell \in [M] : \mathsf{PPRF.Eval}(k_*, M, N+1, \ell) + \mathsf{CRS}_\ell = N+1\}\,.$$
- Pick a random $\ell \in L$, compute $k_p \leftarrow \mathsf{PPRF.Punc}(k_*, M, N+1, \ell)$, and output k_p.

$\mathsf{EvalAll}_0(k_0 = (k_*, N, \hat{\mathbb{G}}, \hat{\mathbb{G}}'), \mathsf{CRS} \in \mathbb{Z}_{N+1}^M)$:

- For every $\alpha \in [N]$, simultaneously compute
$$Y_\alpha \leftarrow |\{\ell \in [M] : \mathsf{PPRF.Eval}(k_*, M, N+1, \ell) + \mathsf{CRS}_\ell = \alpha\}|\,.$$
- Output $Y = (Y_\alpha)_{\alpha \in [N]}$.

$\mathsf{EvalAll}_1(k_p, \mathsf{CRS} \in \mathbb{Z}_{N+1}^M)$:

- For every $\alpha \in [N]$, simultaneously compute
$$Y_\alpha \leftarrow (-\,|\{\ell \in [M] : \mathsf{PPRF.PuncEval}(k_p, \ell) + \mathsf{CRS}_\ell = \alpha\}|)\,.$$
- Output $Y = (Y_\alpha)_{\alpha \in [N]}$.

Figure 4: A modification of the PDPF in Figure 1 in the CRS model. As before, the PDPF is for point functions with output group $\mathbb{G} = \mathbb{Z}$, $\beta \in \{0, 1\}$, and domain size N. Here M is a parameter corresponding to the input space of the PPRF.

4. $k_p \leftarrow \mathsf{Punc}(k, x)$
5. Output (CRS, k_p)

The distribution with an ideal PPRF, in which the punctured point x is completely hidden, and all other points are independent and uniform on $\mathbb{Z}_N$, denoted by $Ideal_\alpha$, is as follows:

1. $\mathsf{CRS} \leftarrow \mathbb{Z}_N^M$
2. $x \leftarrow \mathsf{Choose}\,(\mathsf{CRS}, \alpha)$
3. $\forall z : \mathsf{CRS}^*_z = \begin{cases} \mathsf{CRS}_z, & z \neq x \\ \perp, & z = x \end{cases}$
4. Output CRS^*

We want to argue that the view of an adversary of Real_α can be simulated by an adversary only having access to Ideal_α. This will in particular imply that for any α, α', and any adversary $\mathcal{A}$ against which the PPRF assumption holds, $\mathrm{Adv}[\mathcal{A}, \mathsf{Real}_\alpha, \mathsf{Real}_{\alpha'}]$ is close to $d(\mathsf{Ideal}_\alpha, \mathsf{Ideal}_{\alpha'})$.

As a final note, while Proposition 7 applies to the statistical experiment where a single index is punctured, an almost identical proof applies to the multi-puncturing case.

Proposition 7. *There exists an algorithm Sim such that, for any α and any adversary $\mathcal{A}$ against which the PPRF assumption holds, we have that* $\mathrm{Adv}[\mathcal{A}, \mathsf{Real}_\alpha, \mathsf{Sim}(\mathsf{Ideal}_\alpha)]$ *is negligible.*

Proof. Because CRS is uniform, Real_α is equivalent to the distribution:

1. $k \leftarrow \{0,1\}^\lambda$
2. $\mathsf{CRS} \leftarrow \mathbb{Z}_N^M$
3. $x \leftarrow \mathsf{Choose}\,(\mathsf{CRS}, \alpha)$
4. $k_p \leftarrow \mathsf{Punc}(k, x)$
5. Output $(\mathsf{CRS} + \mathsf{EvalAll}(k), k_p)$

By the PPRF assumption, because x is chosen independent of k, we have that $\mathsf{EvalAll}(k)_x$ is indistinguishable from a random value $r \leftarrow \mathbb{Z}_N$, given the knowledge of k_p. Therefore, the above distribution is equivalent to the following distribution:

1. $\mathsf{CRS} \leftarrow \mathbb{Z}_N^M$
2. $x \leftarrow \mathsf{Choose}\,(\mathsf{CRS}, \alpha)$
3. $k \leftarrow \{0,1\}^\lambda$
4. $r \leftarrow \mathbb{Z}_N$
5. $\forall z : \tau_z = \begin{cases} \mathsf{CRS}_z + \mathsf{EvalAll}(k)_z, & z \neq x \\ r, & z = x \end{cases}$
6. $k_p \leftarrow \mathsf{Punc}(k, x)$
7. Output (τ, k_p)

Next, we want to formulate the above distribution as a function Sim applied to a distribution Ideal_α. Sim on input CRS^* computes the following:

1. Let x be such that $\mathsf{CRS}^*_x = \bot$
2. $k \leftarrow \{0,1\}^\lambda$
3. $r \leftarrow \mathbb{Z}_N$
4. $\forall z : \tau_z = \begin{cases} \mathsf{CRS}^*_z + \mathsf{EvalAll}(k)_z, & z \neq x \\ r, & z = x \end{cases}$
5. $k_p \leftarrow \mathsf{Punc}(k, x)$
6. Output (τ, k_p)

Since $\mathsf{Sim}(\mathsf{Ideal}_\alpha)$ has the same distribution as above, we conclude that $\mathsf{Real}_\alpha \overset{\epsilon}{\approx} \mathsf{Sim}(\mathsf{Ideal}_\alpha)$, where ϵ is the PPRF security error. □