Programmable Distributed Point Functions

Elette Boyle[1], Niv Gilboa[2], Yuval Ishai[3], and Victor I. Kolobov[3]

[1] IDC Herzliya, Israel & NTT Research, USA
elette.boyle@idc.ac.il
[2] Ben-Gurion University, Israel
gilboan@bgu.ac.il
[3] Technion, Israel
{yuvali,tkolobov}@cs.technion.ac.il

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.

- *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.

Keywords: Distributed Point Function, Puncturable Psuedorandom Function.

1 Introduction

A *distributed point function* (DPF) [28,13] 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[4]) into succinctly described

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

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 [28,13].

The appealing compressing structure of DPF constructions has enabled a wide range of cryptographic applications, ranging from Private Information Retrieval (PIR) [19,28], to anonymous messaging systems [21], secure computation for RAM programs [26] and programs with mixed-mode operations [14,7], and recently Pseudorandom Correlation Generators [8,14,12] 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 [13]. In addition, no significantly new approaches toward construction have emerged since this time.

1.1 Our Results

We present a new approach of 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.

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 [25]. 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,[5] 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 [5], and two-server PIR in an offline/online model recently studied by Corrigan-Gibbs and Kogan [22].

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 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 [4,18,17,37]. 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 [22,39] 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, [39] 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.

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

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 [3]).

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 ($1/\text{poly}$**-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 $|k_1| = \lambda \log(/\epsilon^2)$.*

- *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 $1/p(N)$-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 $f_\alpha : [N] \to \{0,1\}$ where the runtime of key generation, single point evaluation, and full domain evaluation is quasilinear in N, and with online key size* $\text{poly}(\lambda, \log N)$.

A few remarks are in order.

Small domains: Applications and non-applications. Note that the key generation and full evaluation algorithms of our construction run in time linear in N, and as such we are restricted to polynomial-sized domains in order to execute within polynomial time. As an additional point of interest, our techniques do not admit a more efficient single-point evaluation algorithm than a full-domain evaluation. An outstanding open problem in our work is to achieve a construction where the running time of key generation and a single point evaluation is only $\text{poly} \log N$.

For many applications of DPFs, the required parameters are anyway on 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 [30] ran 2-server PIR on $N = 8,000$ Netflix movies.

2. *Private "writing"* applications, such as secure distributed storage [36], voting, and aggregation. This includes Prio-style [20] applications for private collection of aggregate statistics, and Riposte [21], Blinder [1] and Spectrum [35] 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 [12,10] (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 [14]. 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 [14,7]. 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.

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 $97.7\,KB$ key size, and the running time for either key generation or full domain evaluation is $72.1\,\mu s$, while our construction achieves $0.5\,KB$ key size, $548.3\,\mu s$ running time for key generation, and $1.6 sec$ running time for full domain evaluation[6]. 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.

[6] In fact, the naive construction, as mentioned in Section 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 3 for more details.

– 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 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.

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 $N = \text{poly}(\lambda)$ 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 [4,18,17,37].

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* $\text{poly}(\lambda, \log M)$.

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 [26] or mixed-mode operations [14,7], use of pseudorandom correlation generators for secure computation preprocessing [8,14,12], 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 [26], 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,[7] 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* $\text{poly}(\lambda, \log N)$*, 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.*

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

[7] 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.

within secure computation of RAM programs anyway requires 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 [26] or for pseudorandom correlation generators (PCGs) of truth-table correlations [11] and (authenticated) multiplication triples [10]. 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[8].

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 $|k_1| = O\left(\lambda t \log \frac{tN}{\epsilon^2}\right)$ 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 $O(\frac{Nt^2}{\epsilon^2})$.*

Due to space limitations, we defer the full treatment of this optimization to the full version of the paper.

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 [22].

Our construction relies on an underlying tool of Punctuarable Pseudorandom Functions (PPRF) [6,32,15]. 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 [29] tree [6,32,15].

Our construction proceeds roughly as follows. Consider the first party in the programmable DPF. The first (programmable) key of the DPF is simply the

[8] 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 $\ell \cdot \ell^2 = \ell^3$.

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.[9] 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 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 $1/\sqrt{c}$.

[9] 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.

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 λ-wise independent. By combining a "statistical-to-perfect" lemma from [33,27] with a computational hardcore lemma of [34], 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 λ-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 where $\mathbb{G} = \mathbb{Z}$, in which 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 [28,13]). *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 small-domain DPF* $(\mathsf{Gen}, \mathsf{Eval}_0, \mathsf{Eval}_1)$ *is a* programmable DPF*, or PDPF for short, if* Gen *can be decomposed into a pair of algorithms* $\Pi = (\mathsf{Gen}_0, \mathsf{Gen}_1)$ *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* $k_1 \in \{0,1\}^*$*.*

Moreover, we require that k_**, returned by* Gen_0 *as part of* k_0*, is a uniform random string, namely,* $k_* \xleftarrow{\$} \{0,1\}^\lambda$*.*

Since the operation of Gen_0 is fixed, in our PDPF constructions we will omit the description of Gen_0. Moreover, we will not be concerned with its running time or the key length of k_0. Finally, since our construction realizes $\mathsf{EvalAll}$ at essentially the same cost as Eval, we will directly describe the $\mathsf{EvalAll}$ algorithm.

In the full version of the paper 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 [13]. There, the simulator is given "leakage" which is the description of the DPF function class. Simulation takes place by simply generating a key for an arbitrary function in the function class.

2.2 Pseudorandom Generators and Functions

We defer the definitions of PRG, PRF and puncturable PRF (PPRF) to the full version of the paper.

Theorem 3 ((P)PRF from OWFs [6,32,15]). *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,* $\mathsf{PPRF} = (\mathsf{Eval}, \mathsf{Punc}, \mathsf{PuncEval})$*, with input domain* $[M]$ *and output domain* $[N]$*, can be implemented with punctured key length* $|k_p| = \lambda \log_2 M$*, such that* $\mathsf{Eval}, \mathsf{Punc}, \mathsf{PuncEval}$ *make* $(\log_2(M/N) \log_2 N)/2\lambda$ *calls to* G*.*

Furthermore, if Eval *or* $\mathsf{PuncEval}$ *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 $\{1, \ldots, N, \perp\}^M$ 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). Then*

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

We prove the lemma in the full version of the paper.

Next, we state Theorem 4, which constructs a PDPF (Figure 1), restricted to 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 configuration, using the truth table of a PPRF. Compared to Lemma 1, this yields an additive error term which is negligible in λ.

We defer the proofs of Theorem 4 and Theorem 5 to the full version of the paper.

Theorem 4 (Small-domain PDPF with $1/\mathrm{poly}(\lambda, N)$ privacy error). *Suppose that $\mathsf{PPRF} = (\mathsf{PPRF.Eval}, \mathsf{PPRF.Punc}, \mathsf{PPRF.PuncEval})$ is a secure PPRF 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,*

$$\epsilon(\lambda, M, N) = \sqrt{\frac{(N+1)}{M}} + negl(\lambda)$$

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 $O(M)$.

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

- Compute $(s, k_{\mathsf{PPRF}}) = G(k_*)$.
- If $\beta = 1$ then $\alpha' \leftarrow \alpha$, else $\alpha' \leftarrow N + 1$
- Find all indices

$$L \leftarrow \left\{\ell \in [M] : \mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, M, N+1, \ell) + s = \alpha'\right\}.$$

- 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 $(s, k_{\mathsf{PPRF}}) = G(k_*)$.
- For every $\alpha \in [N]$, simultaneously compute

$$Y_\alpha \leftarrow \left|\{\ell \in [M] : \mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, M, N+1, \ell) + s = \alpha\}\right|.$$

- Output $Y = (Y_\alpha)_{\alpha \in [N]}$.

$\mathsf{EvalAll}_1(k_1 = (k_p, s))$:

- For every $\alpha \in [N]$, simultaneously compute

$$Y_\alpha \leftarrow \left(-\left|\{\ell \in [M] : \mathsf{PPRF.PuncEval}(k_p, \ell) + s = \alpha\}\right|\right).$$

- Output $Y = (Y_\alpha)_{\alpha \in [N]}$.

Fig. 1. Small-domain computationally $1/\mathrm{poly}(\lambda, N)$-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.

Theorem 5 (Small-domain PDPF over any payload set $\mathbb{G}'$). *If OWFs exists, there exists a small-domain computationally* $\log|\mathbb{G}'|/\mathrm{poly}(\lambda, N)$*-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* $|k_1| = O(\log|\mathbb{G}'|\lambda(\log\lambda + \log N))$*.*

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 in the full version of the paper, 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. [16, Section 4]).

Lemma 2. *Fix integers $\lambda, w > 0$, a prime p, and let $r, N > 0$ be such that $N = \binom{r+w}{r}$ and $r = O(N^{1/w})$. 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. *$q = O(\lambda^2 N^{1/w})$ and $L = O(p^{w+1}\lambda^{w+1}N^{1+\frac{1}{w}})$.*
2. *C, d are computable in polynomial time.*
3. *For every $\alpha \in [N]$, the random variables $\Delta_1, \ldots, \Delta_q$ are λ-wise independent.*

We prove this lemma in the full version of the paper. 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 λ-degree polynomial evaluations through the desired point (namely, Shamir secret sharing of α).

Next, we show that any small-domain computationally $1/\mathrm{poly}(\lambda, N)$-secure PDPF can be transformed into a small-domain PDPF.

Theorem 6. *Fix integers $\lambda, w > 0$, let $r, N > 0$ be such that $N = \binom{r+w}{r}$ and $r = O(N^{1/w})$, and let $p = \mathrm{poly}(\lambda, N)$ be a prime. Furthermore, let $L = L(w, \lambda, N), q = q(w, \lambda, N)$ be as in Lemma 2. Suppose there exists a small-domain computationally $O(1/L \cdot q)$-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 small-domain computationally secure PDPF for point functions with Abelian output group $\mathbb{Z}_p$, domain size N, and key size $|k_1| = q \cdot K$.*

We give the proof of Theorem 6 in the full version of the paper.

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

Next, we prove the following corollary, in similar vein to how Theorem 5 was derived from Theorem 4. Note, however, that here $\mathbb{G}$ cannot be a general (finite) Abelian group, and we are restricted to $\mathbb{G}$ which is a product of $\mathbb{Z}_p$ for prime p.

Corollary 2. *Fix integers $\lambda, w > 0$, and let $r, N > 0$ be such that $N = \binom{r+w}{r}$ and $r = O(N^{1/w})$. If OWFs exist, there exists a $\left(\log|\mathbb{G}| \cdot 2^{-\Omega(\lambda)}\right)$-secure PDPF for point functions with Abelian output group $\mathbb{G} = \prod_i \mathbb{Z}_{p_i}$, where p_i are primes such that $\sum_i p_i \leq \mathrm{poly}(\lambda, N)$, polynomial domain size N, and key size $|k_1| = O(\log|\mathbb{G}|\lambda^3 N^{1/w}(\log\lambda + \log N))$.*

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 $O(1/L \cdot q)$-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.

Fig. 2. Security amplification via LDC

Theorem 5 and Corollary 2 have the downside that their key length grows multiplicatively with $\log |\mathbb{G}|$. We show in the full version of the paper 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.

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 [6,32,15] (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 [5]. 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 [4,18,17,37].

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 [18].

Definition 3 (Privately Puncturable PRF (1-Key, Selective Security)). *A puncturable PRF* $(\mathsf{Gen}, \mathit{Punc}, \mathsf{Eval}, \mathit{PuncEval})$ *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:*

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

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* $(\mathsf{Gen}, \mathit{Punc}, \mathsf{Eval}, \mathit{PuncEval})$*, 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.*

The proof appears in the full version of the paper.

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 $\text{poly}(\lambda, \log N)$ 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 $(\alpha_1, \ldots, \alpha_q) \leftarrow d(\alpha)$, where $d : [N] \to [L]^q$ is as in Lemma 2 (security amplification).[10]
2. Sample q random PPRF keys: $k_1, \ldots, k_q \leftarrow \{0,1\}^\lambda$.
3. For each $i \in [q]$:

[10] 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.

(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 $K_0 = ((k_1, \Delta_1), \ldots, (k_q, \Delta_q))$ and $K_1 = ((k_1^*, \Delta_1), \ldots, (k_q^*, \Delta_q))$.

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 [9,38] (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 [26] 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 ([9,38]). *Consider the GGM-based PPRF construction of [6,32,15]. 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 $k_i^* = \mathsf{Punc}(k_i, x_i^*)$.*

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 $\alpha \in [N]$.[11]

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 $(k_i^*)_{i\in[q]}$.

[11] 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.

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[26]. As stated, the round complexity of our DPF distributed generation protocol is constant (5 rounds), as opposed to $\log N$ as in [26]. The communication complexity of our distributed Gen is also better than [26], due to the roughly $2\times$ improvement in our key size and an additive communication overhead in [26]. 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 [26] it is $\sim 240\,KB$.

The computational complexity of a data access is better than [26] 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 [26] vs. the M data items in our scheme dominates. In [26] 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.

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 [26] or for pseudorandom correlation generators (PCGs) of truth-table correlations [11] and (authenticated) multiplication triples [10].

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 [31,24]. 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 [11], 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 [23]. 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 [10] or low-communication simulation of RAM programs [26].

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. [24]) 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 [23,24]). 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 [14,7], 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.

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 the full version of the paper 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 3 (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. 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. Abraham, I., Pinkas, B., Yanai, A.: Blinder - scalable, robust anonymous committed broadcast. In: CCS '20. pp. 1233–1252 (2020)
2. Agrawal, N., Shamsabadi, A.S., Kusner, M.J., Gascón, A.: QUOTIENT: two-party secure neural network training and prediction. In: ACM CCS 2019. pp. 1231–1247 (2019)
3. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Lightweight techniques for private heavy hitters pp. 762–776 (2021)
4. Boneh, D., Kim, S., Montgomery, H.W.: Private puncturable prfs from standard lattice assumptions. In: EUROCRYPT 2017. pp. 415–445 (2017)
5. Boneh, D., Lewi, K., Wu, D.J.: Constraining pseudorandom functions privately. In: PKC 2017 (2017)
6. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. pp. 280–300 (2013)
7. Boyle, E., Chandran, N., Gilboa, N., Gupta, D., Ishai, Y., Kumar, N., Rathee, M.: Function secret sharing for mixed-mode and fixed-point secure computation. In: EUROCRYPT 2021, Part II. pp. 871–900 (2021)
8. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector ole. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 896–912 (2018)

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

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.

9. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: 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)
10. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-lpn. In: CRYPTO 2020, Part II. pp. 387–416
11. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: CRYPTO 2019 (2019)
12. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-lpn. In: Annual International Cryptology Conference. pp. 387–416. Springer (2020)
13. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: CCS (2016)
14. Boyle, E., Gilboa, N., Ishai, Y.: Secure computation with preprocessing via function secret sharing. In: Theory of Cryptography Conference. pp. 341–371 (2019)
15. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: PKC 2014. pp. 501–519
16. Boyle, E., Ishai, Y., Pass, R., Wootters, M.: Can we access a database both locally and privately? In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part II. pp. 662–693
17. Brakerski, Z., Tsabary, R., Vaikuntanathan, V., Wee, H.: Private constrained prfs (and more) from LWE. In: TCC 2017, Part I. pp. 264–302

18. Canetti, R., Chen, Y.: Constraint-hiding constrained prfs for nc^1 from LWE. In: EUROCRYPT 2017, Part I. pp. 446–476 (2017)
19. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: Proceedings of IEEE 36th Annual Foundations of Computer Science. pp. 41–50. IEEE (1995)
20. Corrigan-Gibbs, H., Boneh, D.: Prio: Private, robust, and scalable computation of aggregate statistics. In: 14th USENIX symposium on networked systems design and implementation (NSDI 17). pp. 259–282 (2017)
21. Corrigan-Gibbs, H., Boneh, D., Mazières, D.: Riposte: An anonymous messaging system handling millions of users. In: 2015 IEEE Symposium on Security and Privacy. pp. 321–338. IEEE (2015)
22. Corrigan-Gibbs, H., Kogan, D.: Private information retrieval with sublinear online time. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020,Part I (2020)
23. Couteau, G.: A note on the communication complexity of multiparty computation in the correlated randomness model. In: EUROCRYPT 2019. pp. 473–503 (2019)
24. Damgård, I., Nielsen, J.B., Nielsen, M., Ranellucci, S.: The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In: CRYPTO 2017
25. Di Crescenzo, G., Malkin, T., Ostrovsky, R.: Single database private information retrieval implies oblivious transfer. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 122–138. Springer (2000)
26. Doerner, J., Shelat, A.: Scaling oram for secure computation. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 523–535 (2017)
27. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: From probing attacks to noisy leakage. In: EUROCRYPT 2014. pp. 423–440 (2014)
28. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: EUROCRYPT (2014)
29. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM **33**(4), 792–807 (Aug 1986)
30. Gupta, T., Crooks, N., Mulhern, W., Setty, S.T.V., Alvisi, L., Walfish, M.: Scalable and private media consumption with popcorn. In: USENIX (2016)
31. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: TCC 2013 (2013)
32. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: CCS 2013. pp. 669–684
33. Maurer, U.M., Pietrzak, K., Renner, R.: Indistinguishability amplification. In: CRYPTO. pp. 130–149 (2007)
34. Maurer, U.M., Tessaro, S.: A hardcore lemma for computational indistinguishability: Security amplification for arbitrarily weak prgs with optimal stretch. In: TCC 2010. pp. 237–254 (2010)
35. Newman, Z., Servan-Schreiber, S., Devadas, S.: Spectrum: High-bandwidth anonymous broadcast with malicious security. IACR Cryptol. ePrint Arch. p. 325 (2021)
36. Ostrovsky, R., Shoup, V.: Private information storage (extended abstract). In: STOC 1997. pp. 294–303 (1997)
37. Peikert, C., Shiehian, S.: Privately constraining and programming prfs, the LWE way. In: PKC 2018, Part II. pp. 675–701 (2018)
38. Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed vector-ole: Improved constructions and implementation. In: ACM CCS. pp. 1055–1072 (2019)
39. Shi, E., Aqeel, W., Chandrasekaran, B., Maggs, B.M.: Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In: CRYPTO. pp. 641–669 (2021)