Oblivious Transfer with Constant Computational Overhead

Elette Boyle[1], Geoffroy Couteau[2], Niv Gilboa[3], Yuval Ishai[4], Lisa Kohl[5], Nicolas Resch[6], and Peter Scholl[7]

[1] IDC Herzliya and NTT Research
[2] IRIF
[3] Ben-Gurion University
[4] Technion
[5] Cryptology Group, CWI Amsterdam
[6] University of Amsterdam
[7] Aarhus University

Abstract. The *computational overhead* of a cryptographic task is the asymptotic ratio between the computational cost of securely realizing the task and that of realizing the task with no security at all.

Ishai, Kushilevitz, Ostrovsky, and Sahai (STOC 2008) showed that secure two-party computation of Boolean circuits can be realized with *constant* computational overhead, independent of the desired level of security, assuming the existence of an oblivious transfer (OT) protocol and a local pseudorandom generator (PRG). However, this only applies to the case of semi-honest parties. A central open question in the area is the possibility of a similar result for *malicious* parties. This question is open even for the simpler task of securely realizing many instances of a constant-size function, such as OT of bits.

We settle the question in the affirmative for the case of OT, assuming: (1) a standard OT protocol, (2) a slightly stronger "correlation-robust" variant of a local PRG, and (3) a standard sparse variant of the Learning Parity with Noise (LPN) assumption. An optimized version of our construction requires fewer than 100 bit operations per party per bit-OT. For 128-bit security, this improves over the best previous protocols by 1-2 orders of magnitude.

We achieve this by constructing a constant-overhead *pseudorandom correlation generator* (PCG) for the bit-OT correlation. Such a PCG generates N pseudorandom instances of bit-OT by locally expanding short, correlated seeds. As a result, we get an end-to-end protocol for generating N pseudorandom instances of bit-OT with $o(N)$ communication, $O(N)$ computation, and security that scales sub-exponentially with N.

Finally, we present applications of our main result to realizing other secure computation tasks with constant computational overhead. These include protocols for general circuits with a relaxed notion of security against malicious parties, protocols for realizing N instances of natural constant-size functions, and reducing the main open question to a potentially simpler question about fault-tolerant computation.

Table of Contents

1 Introduction

A dream goal in cryptography is obtaining security "for free," without any slowdown. How close can we get to this goal in the context of secure computation?

A theoretical study of this question was initiated in the work of Ishai, Kushilevitz, Ostrovsky, and Sahai [IKOS08] (IKOS). For secure two-party computation of Boolean circuits, they showed that it is possible to achieve *constant computational overhead* under plausible cryptographic assumptions. Concretely, there is a multiplicative constant c, independent of the desired security level, such that every sufficiently big Boolean circuit of size N can be securely evaluated by two parties which are implemented by Boolean circuits of size cN.[1] This means that the *amortized* slowdown factor can be independent of the security level.[2]

The IKOS protocol combines a technique of Beaver [Bea96] with a *local PRG* [Gol00, MST03a, AIK04], namely a pseudorandom generator $G : \{0,1\}^\kappa \to \{0,1\}^{n(\kappa)}$ that has polynomial stretch ($n = \Omega(\kappa^d)$ for some $d > 1$) and such that every output bit of G depends on a constant number of input bits. While the existence of such local PRGs was considered quite speculative at the time, it is now widely accepted as a standard cryptographic assumption.

A major limitation of the IKOS protocol is that its security is restricted to the case of semi-honest parties. The possibility of a similar result for *malicious* parties was the main question left open by [IKOS08]. In spite of significant progress on this and related problems, including constant-overhead protocols for *arithmetic* circuits over large fields [ADI$^+$17, BCG$^+$17a], a solution to the above main question is still elusive; see [dCHI$^+$22, JH22] for a survey of related work. The question is open even for simpler tasks, such as computing N instances of a nontrivial constant-size function. To make things worse, strong cryptographic primitives such as indistinguishability obfuscation do not seem helpful. In fact, even entirely heuristic solutions are not currently known. Our work is motivated by the goal of solving useful special cases of this central open question.

The Overhead of Oblivious Transfer. A common framework toward secure computation, including the protocol of IKOS, follows a two-phase approach: first run an input-independent preprocessing protocol for secure distributed generation of useful correlated secret randomness, and then consume these correlations within an online protocol that performs a secure computation on the inputs [Bea92]. An important example is the random *oblivious transfer* (OT) correlation,[3] in which Alice and Bob receive (s_0, s_1) and (b, s_b) respectively, where s_0, s_1, b are random bits. Given $2N$ independent instances of this OT correlation, Alice and Bob can evaluate any Boolean circuit with N gates (excluding "free" XOR and NOT gates) on their inputs, with perfect semi-honest security, by each sending 2 bits and performing a small constant number of bit operations per gate [GMW87, Gol09]. Indeed, IKOS protocol obtains constant-overhead general secure two-party computation precisely by achieving this goal for generation of random bit-OTs.

Generating random bit-OTs with *malicious* security, however, is much more challenging. In particular, the IKOS protocol is not secure in this setting [4] The best known solutions incur polylogarithmic computational overhead [DIK10, dCHI$^+$22]. A natural approach for improvement would be to follow the "GMW-paradigm" [GMW87], applying zero-knowledge proofs to enforce honest behavior in the IKOS protocol. However, the existence of such proofs with constant computational overhead for the satisfiability of Boolean circuits is also wide open: Even there, the best known solutions have polylogarithmic overhead [DIK10]. A number of works developed special-purpose cut-and-choose techniques for protecting efficient OT extension protocols against malicious parties with a very low overhead [KOS15, ALSZ17, Roy22]. However, these techniques are inherently tied to string-OTs whose length is proportional to a security parameter, and seem to require (at least) a polylogarithmic computational overhead when adapted to the case of bit-OT. Part of the challenge of protecting "traditional" OT generation protocols against malicious adversaries is that the underlying semi-honest protocols require $\Omega(N)$ communication for generating N OT correlation instances, which must somehow be checked or verified.

[1] Here the default security requirement is that any $\mathsf{poly}(N)$-time adversary can only obtain a $\mathsf{negl}(N)$ advantage. Alternatively, using a separate security parameter λ, the cN bound holds when N is sufficiently (but polynomially) larger than λ.

[2] See Section 2.1 for more details on our specific cost model. Briefly, functions and protocols are implemented as bounded fan-in Boolean circuits, and the computational cost is the number of gates. For *concrete* computational costs, we allow any bit operation over *two-bit* inputs.

[3] In this work, OT refers by default to bit-OT, namely oblivious transfer of pairs of *bits*. However, as discussed below (cf. Section 6), our results apply to most other natural flavors of OT.

[4] See Appendix A for an explicit attack.

Pseudorandom Correlation Generators. A recent alternative approach to OT generation is via the tool of *pseudorandom correlation generators* (PCG), put forth in [BCG+17b, BCGI18, BCG+19b]. The PCG approach enables fast generation of short correlated seeds, of length $o(N)$, that can be locally expanded without interaction to N instances of OT (or other) correlations. Unlike the traditional protocols from above, the structure of PCG-based protocols directly gives rise to secure computation of N pseudorandom OT correlations with *sublinear* $o(N)$ communication cost. This is an appealing feature, not only as a concrete efficiency benefit (indeed, communication costs often form the practical efficiency bottleneck), but also as a promising starting point for obtaining *malicious* security with low overhead. Indeed, since the local expansion from short PCG seed to long OT output is deterministic, it suffices to ensure that the short seeds be generated correctly, reducing the malicious-security problem to an instance of sublinear size.

However, existing PCG constructions do not yet suffice for our goal. While the communication cost of PCG-based protocols is sublinear in N, the required *computation* costs are quite high. In existing constructions [BCG+19b, BCG+19a, YWL+20, CRR21, BCG+22], even just the *amortized* cost of generating each final bit-OT correlation (corresponding to simply 2 output bits per party) requires generating security-parameter many pseudorandom bits, and then hashing them down.

1.1 Our Results

We present new constructions of pseudorandom correlation generators for N instances of the bit-OT correlation, which not only have sublinear communication in N but also achieve *constant* computational overhead. As a direct consequence, we obtain the first constant-overhead protocol for realizing N instances of bit-OT with security against *malicious* parties. As we further discuss below, this result extends beyond OT to other natural secure computation tasks.

Theorem 1 (Constant-overhead PCG for OT, informal). *Suppose that the following assumptions hold:*

- *There is a local PRG with an additional "correlation robustness" property;*
- *There are sparse generating matrices of codes for which Learning Parity with Noise is hard.*

Then, there is a pseudorandom correlation generator for the bit-OT correlation, with polynomial stretch, where the local expansion function PCG.Expand *has constant computational overhead.*

In fact, we present two variants of this main result: one based on a *primal-LPN* assumption, which has better amortized cost but a small (sub-quadratic) stretch, and one based on *dual-LPN* that can achieve an arbitrary polynomial stretch at the cost of a slightly increased (constant) overhead.

By applying a general-purpose secure computation protocol to distribute the PCG seed generation, we obtain the following corollary.

Corollary 2 (Constant-overhead malicious OT, informal). *Assuming the existence of a standard OT protocol along with the assumptions of Theorem 1, there exists a two-party protocol for realizing N instances of bit-OT with security against malicious parties and a constant computational overhead.*

About the Assumptions. Our protocols require three types of assumptions: (1) the (necessary) existence of standard OT; (2) a slight strengthening of local PRGs that we refer to as *correlation robustness*; and (3) a "sparse" form of the Learning Parity with Noise (LPN) assumption.

As discussed above, local PRGs (more concretely, PRGs with constant locality and polynomial stretch) were already used in the IKOS protocol [IKOS08]. A well-known candidate is Goldreich's PRG [Gol00], where significant study has gone toward proving resilience against classes of attacks for particular choices of output predicates P_i [BQ09, App12, OW14, CEMT14, App15, ABR16, AL16, LV17, BCG+17b, CDM+18]. Correlation robustness of a PRG $G : \{0,1\}^\kappa \to \{0,1\}^N$ requires that for any choice of offsets $\Delta_1, \ldots, \Delta_N \in \{0,1\}^\kappa$, the output $(P_1(x \oplus \Delta_1), \ldots, P_N(x \oplus \Delta_N)) \in \{0,1\}^N$ appears pseudorandom for random x. For a local PRG, this corresponds to fixed xor-shifts of the corresponding output local predicates. In Section C, we investigate the potential correlation robustness of the Goldreich local PRG construction, demonstrating that "good" properties of PRG output predicates P_i are *preserved* under fixed xor-shift. In turn, we conclude that the same classes of attacks can be ruled out for correlation robustness of the PRG as well as for standard pseudorandomness.

"Sparse" LPN, first put forth by Alekhnovich [Ale03], corresponds to a form of LPN whose code generator matrix (i.e., coefficients of noisy linear equations) has constant row sparsity. The assumption states that the mapping $(\vec{s}, \vec{e}) \mapsto \mathbf{G} \cdot \vec{s} + \vec{e}$ is a PRG, where $\vec{s}$ is a short uniform seed of length n, $\mathbf{G} \in \{0,1\}^{N \times n}$ is a suitably chosen sparse matrix and $\vec{e}$ is a noise vector of weight $t \ll N$. In such a scenario we can have polynomial stretch (i.e., both n, t are at most $N^{1-\epsilon}$ for some $\epsilon > 0$) but the stretch is fairly limited.

We also consider the dual variant of LPN directly, where the seed is viewed as a length M error vector $\vec{e}$ and the mapping sends $\vec{e} \mapsto \mathbf{H} \cdot \vec{e}$ for a suitably chosen $\mathbf{H} \in \{0,1\}^{N \times M}$ with $M > N$. By choosing $\mathbf{H}$ to have a *repeat-accumulate* structure, we get desirable efficiency properties (analogous to the efficiency the sparsity of $\mathbf{G}$ earns us in the primal case) while allowing for arbitrary polynomial stretch.

To evaluate the plausibility of our LPN-assumptions we follow the *linear tests*-framework, which was implicit in [BCG+20, CRR21] and made explicit in [BCG+22]. Briefly, this means that we need to verify that the distance of a code related to the matrix is not too small.

About the conclusion. There are several reasons for focusing on the computational overhead of *bit-OT* rather than string-OT or other functionalities. In the semi-honest model, bit-OT is "complete" for general constant-overhead secure computation whereas string-OT is not.[5] We make a meaningful contribution even in the semi-honest model, because of the qualitative improvement in communication cost compared to [IKOS08]. As we will argue below, the completeness of bit-OT partially extends to the malicious model, either by relaxing the security guarantee, by restricting the attention to a broad class of "finite" functionalities, or by reducing the main open question in the area to simpler questions.

Finally, we note that batch-OT serves as a standard benchmark for research on low-overhead cryptography [IKOS09a, IKO+11, DGI+19, OZ22, dCHI+22, BBDP22]. Most of the above works obtained a previously known result under new assumptions. In contrast, our main feasibility result was not previously known under any assumption.

Concrete Amortized Cost. We estimate that, when producing a sufficiently large number N of OTs, our construction based on primal-LPN can have a concrete, amortized cost of 243 bit operations per party, per OT, while achieving sublinear communication complexity. This figure is based on using a PRG with locality 9, which asymptotically is believed to be secure with stretch as large as $\kappa^{2.49}$, and a primal-LPN matrix with row sparsity of $d = 18$. For the dual LPN variant, we can rely on a PRG with locality 5, achieving amortized costs of 91 bit operations per party.

In comparison, with 128 bits of security against malicious parties, the amortized cost of all previous protocols is bigger by 1-2 orders of magnitude, even when using a best-possible implementation of the underlying primitives (e.g., using a local PRG for generating pseudorandom bits). This applies both to protocols with linear communication [IKNP03, KOS15, ALSZ17, Roy22] and to PCG-based protocols with sublinear communication [BCG+19b, BCG+19a, YWL+20, CRR21, BCG+22].

Counting bit operations does not reflect true performance on standard architectures, and in particular does not take into account additional costs such as memory access. However, the extra costs can be amortized by performing many identical computations in parallel. We leave a more thorough investigation of concrete efficiency and further optimizations to future work.

Beyond Oblivious Transfer. While our main result only refers to the specific task of securely realizing N instances of OT, the ubiquity of OT in cryptography makes it relevant to many applications. Even in the strict context of secure computation with constant computational overhead, our results have broader implications to other useful secure computation tasks. We summarize them below.

– *General protocols with relaxed security.* Given our constant-overhead realization of (malicious-secure) OT, one can securely compute *every* Boolean circuit with constant computational overhead by settling for *security up to additive attacks* [GIP+14]. In this relaxed security model, the malicious party can (blindly) choose a subset of the circuit wires to toggle, independently of the honest party's input. While devastating in some applications, such as zero-knowledge proofs, additive attacks can be tolerable in others. This may be the case, for example, when computing functions with long inputs and short outputs, and when the main concern is about the *amount of information* that a malicious party can learn.
– *Leveraging perfect security.* Consider the case of realizing N instances of a "constant-size" functionality f. If f has *perfect* security against *malicious* parties in the *OT-hybrid* model, namely using ideal

[5] Indeed, the main application of string-OT is within protocols based on garbled circuits, which have a super-constant overhead. Note that unlike the case of bit-OT, it is trivial to obtain constant-overhead string-OT via a hybrid approach that combines an arbitrary string-OT with a constant-overhead PRG [IKOS08].

calls to a bit-OT oracle, then our main result implies a constant-overhead protocol in the plain model. While the question of characterizing such f is still open, positive examples include other flavors of OT [BCR86, IPS08], simple noisy channels such as a BSC channel and, more surprisingly, a broad class of functionalities that includes constant-size instances of the millionaire's problem and many others [AP21].

– *Reducing security to fault-tolerance.* Finally, given our constant-overhead realization of the OT-hybrid model, settling the general open question reduces to settling it in this model. This, in turn, reduces to a constant-overhead construction of Boolean *AMD circuits* — randomized circuits that are resilient to additive attacks [GIP+14]. The best known construction of such circuits has polylogarithmic overhead [GIW16].

1.2 Technical Overview

At a high-level, our approach follows the construction of PCGs for random bit-OT via subfield vector oblivious linear evaluation (sVOLE) [BCGI18, BCG+19b]. We first recall their approach, and then explain how to achieve constant overhead.

PCGs for sVOLE from LPN [BCGI18, BCG+19b]. Recall that an sVOLE instance is of the form $(\vec{b}, \vec{z}_0), (x, \vec{z}_1)$, where $x \in \{0,1\}^\kappa, \vec{b} \in \{0,1\}^N, \vec{z}_0 \in (\{0,1\}^\kappa)^N, \vec{z}_1 \in (\{0,1\}^\kappa)^N$ such that $x \cdot \vec{b} = \vec{z}_0 \oplus \vec{z}_1$, where $x \cdot \vec{b} := (b_1 \cdot x, \ldots, b_N \cdot x) \in (\{0,1\}^\kappa)^N$. (Note that typically x is considered as element $x \in \mathbb{F}_{2^\kappa}$, which is where the name subfield VOLE comes from. Here, however, it will be more convenient to think of x as a bitstring $x \in \{0,1\}^\kappa$, since this will later be input to a local PRG.)

The first ingredient of the PCG construction is a pseudorandom generator from the learning parity with noise assumption. Let $n, N \in \mathbb{N}$ with $n < N$. The *primal learning parity with noise assumption* states that, relative to some code generator $\mathbf{C}$ returning matrices in $\{0,1\}^{N \times n}$ and noise distribution $\mathcal{D}$ over $\{0,1\}^N$, $(\mathbf{G}, \mathbf{G} \cdot \vec{s} \oplus \vec{e}) \overset{c}{\approx} (\mathbf{G}, \vec{b})$, where $\mathbf{G} \leftarrow \mathbf{C}, \vec{s} \overset{\$}{\leftarrow} \{0,1\}^n, \vec{e} \leftarrow \mathcal{D}$ and $\vec{b} \overset{\$}{\leftarrow} \{0,1\}^N$. Here, we consider noise distributions $\mathcal{D}$ that return t-sparse vectors, i.e., vectors containing at most t non-zero entries.

The second ingredient is a (known-index) function secret sharing (FSS) scheme to generate a succinct secret sharing of $x \cdot \vec{e}$, where $x \in \{0,1\}^\kappa$ as above, and $\vec{e}$ is a t-sparse noise vector. Roughly, a function secret sharing scheme consists of a tuple of algorithms $(\mathsf{Setup}, \mathsf{FullEval})$, such that $\mathsf{Setup}(1^\lambda, \hat{v})$ (where $\hat{v}$ is the succinct representation of the vector $\vec{v} = x \cdot \vec{e}$) returns a tuple of succinct (i.e., polynonomial in the size of $\hat{v}$) keys (K_0, K_1) and $\mathsf{FullEval}(\sigma, K_\sigma)$ returns a vector $\vec{y}_\sigma \in (\{0,1\}^\kappa)^N$ such that $\vec{y}_0 \oplus \vec{y}_1 = \vec{v}$. The security requirement states, essentially, that even given $\vec{y}_b$ for either $b = 0$ or $b = 1$, one cannot derive any information about $\vec{v}$.

Function secret sharing schemes for so-called t-sparse point functions are known to exist from one-way functions [GI14, BGI15]. Further, as observed in [SGRR19, BCG+19a] for the purpose of constructing PCGs for sVOLE a so-called *known-index* FSS scheme is sufficient, where one party learns the positions of the non-zero entries. Known-index FSS for point functions are implied by simpler constructions of puncturable pseudorandom functions [KPTZ13, BW13, BGI14].

Now, given these two ingredients, the PCG construction for sVOLE can be obtained as follows:

– Sample $x \overset{\$}{\leftarrow} \{0,1\}^\kappa$ as input for P_1.
– Sample $\vec{s} \overset{\$}{\leftarrow} \{0,1\}^n$, $\vec{e} \overset{\$}{\leftarrow} \mathcal{D}$ and give $\vec{s}$, as well as a succinct description of $\vec{e}$ to P_0, who can then compute $\vec{b} := \mathbf{G} \cdot \vec{s} \oplus \vec{e}$.
– Generate a succinct secret sharing of $x \cdot \vec{b}$ as follows:
  1. Generate additive secret shares $\vec{r}_0, \vec{r}_1$ such that $\vec{r}_0 \oplus \vec{r}_1 = x \cdot \vec{s} \in (\{0,1\}^\kappa)^n$.
  2. Generate function secret shares $(K_0, K_1) \leftarrow \mathsf{Setup}(1^\lambda, \hat{v})$, where $\vec{v} := x \cdot \vec{e}$.

By the correctness of the FSS and linearity of the code, it now holds

$$x \cdot \vec{b} = x \cdot (\mathbf{G} \cdot \vec{s} \oplus \vec{e}) = \mathbf{G} \cdot (\vec{r}_0 \oplus \vec{r}_1) \oplus \mathsf{Eval}(0, K_0) \oplus \mathsf{Eval}(1, K_1) = \vec{z}_0 \oplus \vec{z}_1,$$

where $\vec{z}_\sigma := \mathbf{G} \cdot \vec{r}_\sigma + \mathsf{FullEval}(\sigma, K_\sigma)$ for $\sigma \in \{0,1\}$.

From sVOLE to bit-OT. The transformation from sVOLE to bit-OT follows the strategy of [IKNP03]. Namely, an instance of N-dimensional sVOLE can be considered as N instances of correlated string-OT with offset x as follows. The vector $\vec{b}$ corresponds to the choice vector of the "receiver" P_0. Further, for each entry b_i, the receiver obtains $z_{0,i} = z_{1,i} \oplus b_i \cdot x$, i.e., either the bit-string $z_{1,i} \in \{0,1\}^\kappa$ or the bit-string

$z_{1,i} \oplus x \in \{0,1\}^\kappa$ held by the "sender" P_1. These correlations can be removed by applying a correlation-robust hash function $H\colon \{0,1\}^\kappa \to \{0,1\}$. Roughly, a correlation-robust hash function has the property that applied to values related by an (adversarially chosen) Δ, the outputs are indistinguishable from the output on uncorrelated values. With this, the j-th bit OT can be obtained as

$$\big(b_j, H(z_{0,j})\big), \big(H(z_{0,j}), H(z_{0,j} \oplus x)\big).$$

Choosing κ, n, t such that $\kappa \cdot n + t \cdot \log N \in N^{1-\epsilon}$ for some $\epsilon > 0$, the above PCG construction allows to obtain N bit-OTs with communication $o(N)$. On the negative side, it does not achieve constant computational-overhead. The most crucial reason for this is that the sVOLE instance itself introduces an overly large overhead: for each bit-OT, the above transformation requires one to hash κ-bits, introducing a factor κ-overhead (even if all other building blocks are assumed to be constant time). Note that in the above construction it is essential that κ is large, since otherwise a corrupt receiver could guess x and thereby violate the security of the OT.

Towards PCGs for bit-OT with constant overhead. The central idea of this work is to replace the correlation-robust hash function H by a *local pseudorandom generator* G. More precisely, recall that a local PRG is of the form

$$G(x) = P_1(\pi_1(x)) \| \dots \| P_N(\pi_N(x)),$$

where each π_i projects x to an ℓ-sized subset of its coordinates, and $P_i\colon \{0,1\}^\ell \to \{0,1\}$ is a predicate.

Given a local PRG with constant locality ℓ, we can obtain N bit-OTs from an sVOLE instance $x \cdot \vec{b} = \vec{z}_0 \oplus \vec{z}_1$ as

$$\left(b_j, P_j(\pi_j(z_{0,j}))\right), \left(P_j(\pi_j(z_{1,j})), P_j(\pi_j(z_{1,j} \oplus x))\right).$$

In other words, in the j-th bit-OT instance, we replace H by $P_j \circ \pi_j$. Now, it can be shown that if the PRG G additionally satisfies a form of correlation robustness[6], then replacing the correlation-robust hash function by a local PRG preserves correctness and security of the PCG for bit-OT.

This observation does not yet suffice to achieve constant overhead, since the starting point is still an instance of sVOLE with vectors in $(\{0,1\}^\kappa)^N$. Observe though that the parties actually do not need to generate $\vec{z}_0, \vec{z}_1 \in (\{0,1\}^\kappa)^N$, such that $\vec{z}_0 \oplus \vec{z}_1 = x \cdot \vec{b}$. Instead, it suffices to generate $\vec{v}_0, \vec{v}_1 \in (\{0,1\}^\ell)^N$, such that

$$\pi_j(x) \cdot b_j = v_{0,j} \oplus v_{1,j}$$

for all $j \in [N]$, where $\ell \in \mathbb{N}$ is constant. The above generation of bit-OTs can then be simplified as

$$\big(b_j, P_j(v_{0,j})\big), \big(P_j(v_{j,1}), P_j(v_{j,1} \oplus \pi_j(x))\big),$$

where equality holds since the projection functions are linear. We will refer to this correlation as *projected-payload sVOLE* in the following. It remains to discuss how to generate a projected payload sVOLE PCG with constant overhead.

Projected payload sVOLE via primal LPN. Recall that we need to generate compressed secret sharings of $x \cdot \vec{b} = \mathbf{G} \cdot (x \cdot \vec{s}) + x \cdot \vec{e}$. Towards constant overhead, we first replace $\mathbf{G}$ by a sparse matrix, for which each row only contains a constant number d of non-zero entries. By an Alekhnovich variant of the LPN assumption [Ale03], the resulting $\vec{b}$ is still computationally hard to distinguish from random (given a suitable choice of parameters). This allows P_0 to compute $\vec{b} = \mathbf{G} \cdot \vec{s} + \vec{e}$ with constant overhead $O(d \cdot N + t \cdot \log N)$.

Again, we generate secret shares $\vec{r}_0, \vec{r}_1$ such that $\vec{r}_0 \oplus \vec{r}_1 = x \cdot s \in (\{0,1\}^\kappa)^n$. If $\kappa \cdot n < N$, these shares have size $< N$ as required. For expansion, note that for each $j \in [N]$ it is sufficient to compute

$$\pi_j(x) \cdot \mathbf{G}_j \cdot \vec{s} = \mathbf{G}_j \cdot (\pi_j(x) \cdot \vec{s}) = \mathbf{G} \cdot (\mathit{\Pi}_j(\vec{r}_0) \oplus \mathit{\Pi}_j(\vec{r}_1)),$$

where $\mathbf{G}_j$ is the j-th row of $\mathbf{G}$, and $\mathit{\Pi}_j\colon (\{0,1\}^\kappa)^N \to (\{0,1\}^\ell)^N$ is obtained by applying π_j componentwise. Overall, expansion requires $d \cdot N$ operations.

Finally, recall that by above considerations it is left to generate secret shares $\vec{v}_0, \vec{v}_1 \in (\{0,1\}^\ell)^N$ such that $v_{0,j} \oplus v_{1,j} = \pi_j(x) \cdot e_j$.

We can do this with constant overhead by relying on LPN with regular noise, i.e., where $\vec{e}$ is split into N/t intervals, each containing exactly one non-zero noise coordinate. For the corresponding class

[6] Namely, we require $\{P_j(\Delta_j \oplus \pi_j(x))\}_{j \in N}$ is indistinguishable from random, where $\Delta_1, \dots, \Delta_N$ are pseudorandom with seed known to the adversary.

$\{\pi_j(x) \cdot e_j\}_{j\in[N]}$, one can achieve a known-index FSS with constant overhead by using the puncturable PRF construction of [KPTZ13, BW13, BGI14] together with an observation in [BCGI18], which allows to remove a factor-λ overhead. This only requires a length-doubling PRG, which can be instantiated with constant overhead using the same PRG with constant locality as before.

Projected-payload sVOLE via dual LPN. The above construction suffices for constant-overhead OT, although the PCG is limited to subquadratic stretch. We can obtain arbitrary polynomial stretch by generating $\vec{b}$ via *dual LPN*, i.e., as $\vec{b} = \mathbf{H} \cdot \vec{e}$, where $\mathbf{H} \in \{0,1\}^{N\times M}$, $\vec{e} \in \{0,1\}^M$ (where $M = d \cdot N$). To achieve constant locality, we choose $\mathbf{H}$ such that $\mathbf{H} = \mathbf{B} \cdot \mathbf{A}$, where $\mathbf{A}$ is an accumulator matrix (i.e., an all-one lower-triangular matrix) and $\mathbf{B}$ has only a constant number d of non-zero entries in each column. The security is based on a "repeat-accumulate" variant of LPN, which is analogous to the expand-accumulate LPN assumption that appeared recently [BCG+22].

In this case, for $\vec{b} = \mathbf{H} \cdot \vec{e}$, the goal is now to generate compressed secret shares of $(b_1 \cdot \pi_1(x), b_2 \cdot \pi_2(x), \ldots, b_N \cdot \pi_N(x))$. Fortunately for us, we know how to share $\vec{a} := \mathbf{A} \cdot \vec{e}$ in a compressed manner: $\vec{a}$ is a multi-interval noise vector, and so we can share it using function secret-sharing for multi-interval functions. More precisely, by a t-multi-interval noise vector we mean a vector in which there are at most t coordinates $i \geq 2$ for which the i-th coordinate differs from the $(i-1)$-st. However, as $\vec{b} = \mathbf{B} \cdot \vec{a}$, we need to work a bit harder.

Fortunately, recall that each row of $\mathbf{B}$ only has d nonzero entries, and d is a *constant*. Let $S_j \subseteq [M]$ be such that $b_j = \bigoplus_{i\in S_j} a_i$. To get shares of $b_j\pi_j(x)$, it suffices to secret share $a_i \cdot \pi_j(x)$ for exactly these d choices of $i \in S_j$. We thereby get secret shares $\vec{v}^0, \vec{v}^1 \in (\{0,1\}^\ell)^M$. In particular, to obtain an *additive* secret-sharing of $b_j \cdot \pi_j(x)$ for $j \in [N]$, each party $\sigma \in \{0,1\}$ just needs to locally compute $\bigoplus_{i\in S_j} v_i^\sigma$. That is, $\bigoplus_{i\in S_j} v_i^0 \oplus \bigoplus_{i\in S_j} v_i^1 = b_j \cdot \pi_j(x)$.

To distribute the shares $\vec{v}_0, \vec{v}_1$, we introduce an FSS variant called *projected-payload distributed comparison function*, which optimizes for the fact that, at each index j, only the projection $\pi_j(x)$ is multiplied with the bits a_i of the interval vector for $i \in S_j$. This is contrasted with a standard distributed comparison function, where the whole of the κ-bit x is multiplied for every a_i.

We show how to build projected-payload DCF with constant overhead, by carefully combining a standard (known-index) DCF with a DPF. In a nutshell, we use a DPF and DCF which both correspond to a truncated binary tree, with N/κ leaves instead of N. The DCF is set to give out shares of the full payload x for indices i such that $\lceil i/\kappa \rceil < \alpha'$, where $\alpha' = \lceil \alpha/\kappa \rceil$, and shares of 0 otherwise. Note that this already allows the parties to obtain shares of the *projected* evaluations $a_i \cdot \pi_i(x)$, for all $i \in [N]$ except those whose prefix is α'. To correct for the indices with prefix α', we give out an $\ell\kappa$-bit correction word, which is masked using the missing expanded output of the DPF, and allows the party who knows α to correct its outputs to the right value.

2 Preliminaries

2.1 Computational Model and Cost Measure

Computational cost. Similarly to prior works [IKOS08, AM13, AHI+17, BISW18, dCHI+22, RR22, JH22, FLY22, CLY22], we assume that functions and protocols are implemented using Boolean circuits with bounded fan-in gates. Computational cost is then measured by the circuit size, namely the number of gates. Note that this cost measure is robust to the exact fan-in or the type of gates used, which can only change the cost by a constant multiplicative factor. This should be contrasted with counting atomic operations in more liberal computational models, such as arithmetic circuits or RAM programs, which are more sensitive to model variations such as the underlying ring or the allowable word size. See [Spi95] for discussion.

Concrete cost. When we refer to *concrete* computational costs, we count the number of *bit operations* by considering the size of a circuit over the full binary basis, namely where a gate can compute any mapping from two bits to one bit. For instance, the concrete computational cost of the predicate $P_5 =: (x_1 \wedge x_2) \oplus x_3 \oplus x_4 \oplus x_5$ is 4. This is a standard concrete cost measure in complexity theory.

Setup. When considering the computational cost of cryptographic tasks, we allow a one-time PPT setup that given a security parameter 1^λ and a task description, outputs a circuit implementation for the task. For instance, for the task of generating N instances of random bit-OT, the task description is 1^N and the implementation includes circuits computing the next-message functions of the protocol.

Since the setup cost is amortized away, we do not consider its complexity except for requiring it to run in polynomial time. The setup will typically need to generate constant-degree bipartite expander graphs in which one side is polynomially bigger than the other. A recent PPT construction of such graphs with negligible failure probability was given in [AK19]. Alternatively, the failure probability can be eliminated assuming the conjectured existence of explicit unbalanced expanders or similar combinatorial objects; see, e.g., [IKOS08, ADI+17] for discussion. Under this assumption, the setup required by our protocols can be implemented in deterministic polynomial time.

Computational overhead. We will be interested in minimizing the amortized computational cost of a task of size N (e.g., N instances of random bit-OT), when N tends to infinity. Here we allow $N = N(\lambda)$ to be an arbitrarily big polynomial in the security parameter, effectively ignoring lower order additive terms that may depend polynomially on λ and sublinearly on N.[7] We say that the implementation has *computational overhead (at most)* $c = c(\lambda)$ if there is a polynomial $N = N(\lambda)$ such that ratio between the implementation cost and $N(\lambda)$ is at most $c(\lambda)$ for all sufficiently large λ. We say that the implementation has *constant computational overhead* if $c(\lambda) = O(1)$.

As discussed in [IKOS08], a cleaner alternative is to use N both as a size parameter and a security parameter, similarly to textbook definitions of basic cryptographic primitives. (Here security means that every $\mathsf{poly}(N)$-size adversary only has a $\mathsf{negl}(N)$ advantage.) The separation between the two parameters serves to simplify the presentation and give a better sense of concrete efficiency.

Cost of pseudorandomness. Sometimes, it will be convenient to refer to the amortized cost of outputting n pseudorandom bits from a PRG seed. We write this as $C_{\mathsf{prg}}(n)$.

Note that using local PRGs, we have $C_{\mathsf{prg}}(n) = O(n)$, where the best concrete candidate (using the P_5 predicate described above) has cost $C_{\mathsf{prg}}(n) = 4n$. To analyze efficiency on modern CPUs, it can be useful to measure this cost separately due to the prevalence of built-in hardware support for AES. However, note that for large values of n, a "bitsliced" implementation of a local PRG (evaluating many PRG copies in parallel using *bitwise* AND and XOR operations) may have better performance, at the expense of using a much bigger seed.

2.2 Correlation Robust Local PRGs

In this section we recall local pseudorandom generators and introduce the notion of *correlation-robustness* in the context of local PRGs.

Definition 3 (Pseudorandom generator). *Let $\kappa = \kappa(\lambda), N = N(\lambda) \in \mathbb{N}$. We say a family of functions $G = \{G_\lambda \colon \{0,1\}^{\kappa(\lambda)} \to \{0,1\}^{N(\lambda)}\}_{\lambda \in \mathbb{N}}$ is a* pseudorandom generator (PRG)*, if for all polynomial-time non-uniform adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl} : \mathbb{N} \to R_{\geq 0}$ *such that for all* $\lambda \in \mathbb{N}$:

$$\left|\Pr[\mathcal{A}(1^\lambda, G_\lambda(x)) = 1 \mid x \leftarrow \{0,1\}^\kappa] - \Pr[\mathcal{A}(1^\lambda, u) = 1 \mid u \leftarrow \{0,1\}^N]\right| \leq \mathsf{negl}(\lambda).$$

Definition 4 (Subset projection). *Let $\kappa, \ell \in \mathbb{N}$ with $\kappa > \ell$. We say a mapping $\pi \colon \{0,1\}^\kappa \to \{0,1\}^\ell$ is a* subset projection *(or simply* projection*), if there exists a size-ℓ set $S \subset \{1, \ldots, \kappa\}$ such that $\pi(x) = (x_i)_{i \in S}$ for all $x \in \{0,1\}^\kappa$.*

Definition 5 (Local family of functions). *Let $\kappa = \kappa(\lambda), N = N(\lambda), \ell = \ell(\lambda) \in \mathbb{N}$ with $\ell \ll \kappa$ (e.g., $\ell = O(\log \kappa)$ or $\ell = O(1)$). We say a family of functions $G = \{G_\lambda \colon \{0,1\}^{\kappa(\lambda)} \to \{0,1\}^{N(\lambda)}\}_{\lambda \in \mathbb{N}}$ is* ℓ-local *if there exists families of subset projections $\pi_1, \ldots, \pi_{N(\lambda)} \colon \{0,1\}^\kappa \to \{0,1\}^{\ell(\lambda)}$ and families of predicates $P_1, \ldots, P_{N(\lambda)} \colon \{0,1\}^{\ell(\lambda)} \to \{0,1\}$, such that for every $\lambda \in \mathbb{N}$,*

$$G(x) = P_1(\pi_1(x)) \| \ldots \| P_{N(\lambda)}(\pi_{N(\lambda)}(x))$$

for all $x \in \{0,1\}^{\kappa(\lambda)}$. We say G has constant locality *if $\ell \in O(1)$.*

[7] This should be contrasted with a more fine-grained measure of overhead considered in [BIO14, BISW18, dCHI+22], which requires *exponential* security in λ (rather than super-polynomial), measures the overhead with respect to $N + \lambda$, and requires the overhead to apply to all choices of N and λ (e.g., even when $N = \lambda$).

Definition 6 (Δ-shift). *Let $\kappa, N, \ell \in \mathbb{N}$ with $\ell < \kappa$ and let $G\colon \{0,1\}^\kappa \to \{0,1\}^N$ be a ℓ-local function with subset projections $\pi_1, \dots, \pi_N\colon \{0,1\}^\kappa \to \{0,1\}^\ell$ and predicates $P_1, \dots, P_N\colon \{0,1\}^\ell \to \{0,1\}$. For $\Delta = (\Delta_1, \dots, \Delta_N) \in \{0,1\}^N$, we define the Δ-*shift *of G as*

$$G^\Delta(x) = P_1(\pi_1(x) \oplus \Delta_1)\| \dots \|P_N(\pi_N(x) \oplus \Delta_N)$$

for all $x \in \{0,1\}^\kappa$.

Definition 7 (Correlation-robust local PRG). *Let $\kappa = \kappa(\lambda), N = N(\lambda), \ell = \ell(\lambda) \in \mathbb{N}$. Let $G = \{G_\lambda\colon \{0,1\}^{\kappa(\lambda)} \to \{0,1\}^{N(\lambda)}\}_{\lambda\in\mathbb{N}}$ be a family of local functions with ℓ-locality.*

We say that G is a correlation-robust ℓ-local PRG, *if for all polynomial-time non-uniform adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ *such that for all $\lambda \in \mathbb{N}$*

$$\left| \Pr\left[\mathcal{A}_2(\mathsf{st}, y) = 1 \,\middle|\, \begin{array}{c} (\Delta, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda) \\ x \overset{\$}{\leftarrow} \{0,1\}^{\kappa(\lambda)} \\ y = G_\lambda^\Delta(x) \end{array} \right] - \Pr\left[\mathcal{A}_2(\mathsf{st}, y) = 1 \,\middle|\, \begin{array}{c} (\Delta, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda) \\ y \overset{\$}{\leftarrow} \{0,1\}^{N(\lambda)} \end{array} \right] \right|$$
$$\leq \mathsf{negl}(\lambda),$$

where $\Delta \in \{0,1\}^{N(\lambda)}$ and G_λ^Δ is the Δ-shift of G^λ, as defined in Def. 6.

Note that this definition implies the standard definition of pseudorandomness since the adversary can choose $\Delta = 0$.

Note that for our constructions it is actually sufficient to rely on a *weaker* distributional version of correlation-robustness, where the adversary does not have control over Δ, but where it is sampled according to a distribution (for a formal definition see below). For simplicity (and since our analysis in Appendix C suggests no weaknesses for known instantiations even if Δ is adversarially chosen), we will rely on the stronger version in the body of the paper.

In the distributional variant of correlation-robustness, Δ is chosen relative to some distribution $\mathcal{D} = \mathcal{D}_\lambda$ returning tuples of the form (Δ, st), where st is some state that the adversary obtains. Note that to instantiate our construction of constant-overhead OT with this weaker notion of correlation-robustness, the distribution would correspond to a pseudorandom distribution, where st is a random seed used to generate Δ.

Definition 8 (Correlation-robust local PRG relative to $\mathcal{D}$). *Let $\kappa = \kappa(\lambda), N = N(\lambda), \ell = \ell(\lambda) \in \mathbb{N}$. Let $\mathcal{D} = \mathcal{D}_\lambda$ be a distribution returning tuples of the form (Δ, st), where $\Delta \in \{0,1\}^{\kappa(\lambda)}$. Let $G = \{G_\lambda\colon \{0,1\}^{\kappa(\lambda)} \to \{0,1\}^{N(\lambda)}\}_{\lambda\in\mathbb{N}}$ be a family of local functions with ℓ-locality.*

We say that G is a correlation-robust ℓ-local PRG *if for all polynomial-time non-uniform adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ *such that for all $\lambda \in \mathbb{N}$*

$$\left| \Pr\left[\mathcal{A}(\Delta, \mathsf{st}, y) = 1 \,\middle|\, \begin{array}{c} (\Delta, \mathsf{st}) \leftarrow \mathcal{D}_\lambda \\ x \overset{\$}{\leftarrow} \{0,1\}^{\kappa(\lambda)} \\ y = G_\lambda^\Delta(x) \end{array} \right] - \Pr\left[\mathcal{A}(\Delta, \mathsf{st}, y) = 1 \,\middle|\, \begin{array}{c} (\Delta, \mathsf{st}) \leftarrow \mathcal{D}_\lambda \\ y \overset{\$}{\leftarrow} \{0,1\}^{N(\lambda)} \end{array} \right] \right|$$
$$\leq \mathsf{negl}(\lambda),$$

where G_λ^Δ is the Δ-shift of G^λ as defined in Def. 6.

2.3 Learning Parity With Noise

We define the LPN assumption over a ring $\mathcal{R}$ with number of samples N w.r.t. a code generation algorithm $\mathbf{C}$ and a noise distribution $\mathcal{D}$. In the following we state a primal and a dual version of the LPN assumption. Note that we consider LPN and dual-LPN in the bounded sample regime, which is commonly referred to as the *syndrome decoding assumption* in the code-based cryptography literature.

Definition 9 (Primal LPN). *Let $\mathcal{D}(\mathcal{R}) = \{\mathcal{D}_{n,N}(\mathcal{R})\}_{n,N\in\mathbb{N}}$ denote a family of distributions over a ring $\mathcal{R}$, such that for any $n, N \in \mathbb{N}$, $\mathsf{Im}(\mathcal{D}_{n,N}(\mathcal{R})) \subseteq \mathcal{R}^N$. Let $\mathbf{C}$ be a probabilistic code generation algorithm such that $\mathbf{C}(N, n, \mathcal{R})$ outputs a matrix $\mathbf{G} \in \mathcal{R}^{N\times n}$. For dimension $n = n(\lambda)$, number of samples (or block length) $N = N(\lambda)$, and ring $\mathcal{R} = \mathcal{R}(\lambda)$, the (primal) $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-$\mathsf{LPN}(n, N)$ assumption states that*

$$\{(\mathbf{G}, \vec{b}) \mid \mathbf{G} \overset{\$}{\leftarrow} \mathbf{C}(N, n, \mathcal{R}), \vec{e} \overset{\$}{\leftarrow} \mathcal{D}_{n,N}(\mathcal{R}), \vec{s} \overset{\$}{\leftarrow} \mathcal{R}^n, \vec{b} \leftarrow \mathbf{G} \cdot \vec{s} + \vec{e}\}$$
$$\overset{c}{\approx} \{(\mathbf{G}, \vec{b}) \mid \mathbf{G} \overset{\$}{\leftarrow} \mathbf{C}(N, n, \mathcal{R}), \vec{b} \overset{\$}{\leftarrow} \mathcal{R}^N\}$$

Definition 10 (Dual LPN). *Let $\mathcal{D}(\mathcal{R}) = \{\mathcal{D}_{N,M}(\mathcal{R})\}_{n,N\in\mathbb{N}}$ denote a family of efficiently sampleable distributions over a ring $\mathcal{R}$, such that for any $N, M \in \mathbb{N}$, $\mathsf{Im}(\mathcal{D}_{N,M}(\mathcal{R})) \subseteq \mathcal{R}^M$. Let $\mathbf{C}$ be a probabilistic code generation algorithm such that $\mathbf{C}(N, M, \mathcal{R})$ outputs a matrix $\mathbf{H} \in \mathcal{R}^{N\times M}$. For dimension $M = M(\lambda)$, number of samples $N = N(\lambda)$, and ring $\mathcal{R} = \mathcal{R}(\lambda)$, the (dual) $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-$\mathsf{LPN}(N, M)$ assumption states that*

$$\{(\boldsymbol{H}, \vec{b}) \mid \boldsymbol{H} \xleftarrow{\$} \mathbf{C}(N, M, \mathcal{R}), \vec{e} \xleftarrow{\$} \mathcal{D}_{N,M}(\mathcal{R}), \vec{b} \leftarrow \boldsymbol{H} \cdot \vec{e}\}$$
$$\overset{c}{\approx} \{(\boldsymbol{H}, \vec{b}) \mid \boldsymbol{H} \xleftarrow{\$} \mathbf{C}(N, M, \mathcal{R}), \vec{b} \xleftarrow{\$} \mathcal{R}^N\}.$$

If $\mathbf{C}(N, n, \mathcal{R})$ always outputs the same matrix $\mathbf{G} \in \mathcal{R}^{N\times n}$ (in the primal case) or $\mathbf{H} \in \mathcal{R}^{N\times M}$ (in the dual case), we simplify the notation to $(\mathcal{D}, \mathbf{G}, \mathcal{R})$-$\mathsf{LPN}(n, N)$ (in the primal case) or $(\mathcal{D}, \mathbf{H}, \mathcal{R})$-$\mathsf{LPN}(N, M)$ (in the dual case).

Remark 11 (LPN with regular noise). In this work, for the noise distribution we will use $\mathsf{Reg}_t^N(\{0, 1\})$ which outputs a concatenation of t random unit vectors from $\{0, 1\}^{N/t}$. This variant of choosing regular noise was introduced in [AFS03], has been further analysed in [BCGI18] and [HOSS18], and has found applications in the PCG line of work as it significantly improves efficiency [BCGI18,BCG+19b,BCG+19a]. While building on regular noise does not seem to affect security of dual LPN in the parameter regimes considered in the line of work on PCGs, it requires a more careful parameter instantiation for primal LPN.

Note that, somewhat counter-intuitively, LPN with regular noise is trivially broken in the "too-high" noise regime, since for each of the t intervals the corresponding rows of the matrix and entries of $\vec{b}$ can be summed up, resulting in a linear system of t non-noisy equations (note that this can be achieved by flipping the sum of the entries in $\vec{b}$, since we are guaranteed there is exactly one non-zero noise entry in each interval). Further, the Arora-Ge attack [AG11] on LPN with structured noise applies if $n \in O(N^2)$. While these do not affect the parameters of dual-LPN for typical choices of parameters we use (where $n = M - N = O(N)$ and $t \ll N$), this has to be taken into account when choosing the parameters for our primal-LPN instantiation.

LPN-friendliness. In order to develop more efficient protocols, we will consider code generation algorithms that output matrices with useful structure. To determine when the primal/dual-LPN assumption plausibly holds, we follow the recently proposed heuristic of *resilience to linear tests*. As discussed in detail in [BCG+22],[8] essentially all attacks on the LPN problem for our range of parameters involve choosing a nonzero attack vector $\vec{u} \in \{0, 1\}^N \setminus \{\vec{0}\}$ and then computing the dot product $\vec{u}^\top \cdot \vec{b}$, where either $\vec{b} \xleftarrow{\$} \{0, 1\}^N$, or $\vec{b} = \mathbf{G} \cdot \vec{s} + \vec{e}$ in the primal case or $\vec{b} = \mathbf{H} \cdot \vec{e}$ in the dual case. The hope is to detect a noticeable bias in the bit $\vec{u}^\top \cdot \vec{b}$, as in the case $\vec{b}$ is uniform the bit $\vec{u}^\top \cdot \vec{b}$ is perfectly unbiased. Concretely, for a vector $\vec{v} \in \{0, 1\}^N$ and a distribution $\mathcal{D}$ with $\mathsf{Im}(\mathcal{D}) \subseteq \{0, 1\}^N$ we define the *bias* of $\vec{v}$ with respect to $\mathcal{D}$ as

$$\mathsf{bias}_{\vec{v}}(\mathcal{D}) = \left|\mathbb{E}_{\vec{x}\sim\mathcal{D}}[\vec{v}^\top \cdot \vec{x}] - \frac{1}{2}\right| .$$

Concretely, for a vector $\vec{v} \in \{0, 1\}^N$ of weight D we have $\mathsf{bias}_{\vec{v}}(\mathsf{Reg}_t^N) \le (1 - 2(D/t)/(N/t))^t < e^{-2tD/N}$.

For the primal case, to rule out the existence of a good linear test it suffices to show that the code generated by $\mathbf{G}$ has good dual distance. More concretely, letting $\mathsf{HW}(\vec{u})$ denote the number of nonzero entries the vector (its *weight*) it should be that any nonzero vector $\vec{u}$ satisfying $\vec{u}^\top \cdot \mathbf{G} = \vec{0}^\top$ has $\mathsf{HW}(\vec{u}) \ge D$ (say). To see this, note that if $\vec{u}^\top \cdot \mathbf{G} \ne \vec{0}^\top$ then $\vec{u}^\top \cdot (\mathbf{G} \cdot \vec{s})$ will be perfectly unbiased (since $\vec{s} \xleftarrow{\$} \{0, 1\}^n$), so $\vec{u}^\top \cdot \vec{b}$ will be perfectly unbiased. Otherwise $\vec{u}^\top \cdot \vec{b} = \vec{u}^\top \cdot \vec{e}$ whose bias will not be too large assuming both $\mathsf{HW}(\vec{u}) \ge D$ and $\mathsf{HW}(\vec{e}) \ge t$. In particular, once $Dt > \lambda N \ln(2)/2$ the bias will be at most $2^{-\lambda}$.

The dual case is similar: we would like that there is no light vector of the form $\vec{u}^\top \cdot \mathbf{H}$ for $\vec{u} \in \{0, 1\}^N \setminus \{\vec{0}\}$, as if all such vectors $\vec{u}$ satisfy $\mathsf{HW}(\vec{u}^\top \cdot \mathbf{H}) \ge D$ then the bias of $\vec{u}^\top \cdot \mathbf{H} \cdot \vec{e}$ will be at most $e^{-2Dt/M}$, so we can take $Dt > \lambda M \ln(2)/2$ to guarantee bias at most $2^{-\lambda}$.

2.4 Function Secret Sharing

We use several types of function secret sharing for different function classes, including point functions and interval functions. We relax the standard definition [BGI16] by allowing some additional leakage

[8] We refer the interested reader to this work for more details.

given to one of the parties. The leakage will be the point/ interval positions to party P_0. As observed in [SGRR19, BCG+19a], in the context of PCGs for OT and VOLE, allowing this leakage can give rise to more efficient instantiations without hurting security (since P_0 already knows the LPN noise values anyway).

FSS with per-party leakage. Following the syntax of [BGI16], we consider a function family to be defined by a pair $\mathcal{F} = (P_\mathcal{F}, E_\mathcal{F})$, where $P_\mathcal{F}$ is an infinite collection of function descriptions $\hat{f}$ (containing the input domain D_f and range R_f), and $E_\mathcal{F} \colon P_\mathcal{F} \times \{0,1\}^* \to \{0,1\}^*$ is a polynomial-time algorithm defining the function described by $\hat{f}$. More concretely, each $\hat{f} \in P_\mathcal{F}$ describes a corresponding function $f \colon D_f \to R_f$ defined by $f(x) = E_\mathcal{F}(\hat{f}, x)$. In the following, we will typically have $D_f = [N]$ (where $[N] = \{1, \ldots, N\}$), and $R_f = \mathbb{G}$ for some group $\mathbb{G}$.

Note that unlike the original definition, we include the full-domain evaluation algorithm, $\mathsf{FullEval}$, as part of the definition for the following reason. While Eval implies the existence of $\mathsf{FullEval}$ (and vice versa), considering the two independently can give rise to more efficient implementations. If only considering $\mathsf{FullEval}$ for evaluation, we will sometimes write $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{FullEval})$.

Definition 12 (FSS Syntax). *A (2-party) function secret sharing scheme (FSS) is a pair of algorithms* $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{FullEval})$ *with the following syntax:*

– $\mathsf{Setup}(1^\lambda, \hat{f})$ *is a PPT algorithm, which on input of the security parameter 1^λ) and the description of a function $\hat{f} \in \{0,1\}^*$ outputs a tuple of keys (K_0, K_1).*
– $\mathsf{Eval}(\sigma, K_\sigma, x)$ *is a polynomial-time algorithm, which on input of the party index $\sigma \in \{0,1\}$, key K_σ, and input $x \in [N]$, outputs a group element $y_\sigma \in \mathbb{G}$.*
– $\mathsf{FullEval}(\sigma, K_\sigma)$ *is a polynomial-time algorithm, which on input of the party index $\sigma \in \{0,1\}$ and key K_σ, outputs a vector $(\vec{y}_\sigma) \in \mathbb{G}^N$.*

Definition 13 (FSS Security). *Let $\mathcal{F} = (P_\mathcal{F}, E_\mathcal{F})$ be a function family and $\mathsf{Leak}_0, \mathsf{Leak}_1 \colon \{0,1\}^* \to \{0,1\}^*$ be the respective leakage functions. A secure FSS for $\mathcal{F}$ with leakage Leak is a pair $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{FullEval})$ as in Definition 12, satisfying the following:*

– ***Correctness:*** *For all $\hat{f} \in P_\mathcal{F}$ describing $f \colon [N] \to \mathbb{G}$, and every $x \in [N]$, if $(K_0, K_1) \leftarrow \mathsf{Setup}(1^\lambda, \hat{f})$, then*
$$\Pr[\mathsf{Eval}(0, K_0, x) + \mathsf{Eval}(1, K_1, x) = f(x)] = 1 \text{ and}$$
$$\Pr[\mathsf{FullEval}(0, K_0) + \mathsf{FullEval}(1, K_1) = \{f(x)\}_{x \in [N]}] = 1$$
– ***Secrecy:*** *For each $\sigma \in \{0,1\}$, there exists a PPT algorithm Sim such that for every sequence $\hat{f}_1, \hat{f}_2, \ldots$ of polynomial-size function descriptions from $P_\mathcal{F}$, the outputs of the following experiments Real and Ideal are computationally indistinguishable:*
  - $\mathsf{Real}(1^\lambda)$: *Sample $(K_0, K_1) \leftarrow \mathsf{Setup}(1^\lambda, \hat{f}_\lambda)$ and output K_σ.*
  - $\mathsf{Ideal}(1^\lambda)$: *Output $\mathsf{Sim}(1^\lambda, \mathsf{Leak}_\sigma(\hat{f}_\lambda))$.*

In the following, when referring to an FSS, we always assume the FSS to satisfy correctness and secrecy.

Remark 14 (Pseudorandomness of the output shares). In [BGI15] it was shown that for any sufficiently rich function class (including point functions and interval functions considered below), secrecy implies pseudorandomness of the output shares.

Remark 15 (Succinctness). Note that the running time of the Setup algorithm (and therefore the key sizes) are only allowed to depend polynomially on the size of the description $\hat{f}$ of f. We will refer to the computational cost of Setup as $C_{\mathsf{FSS.Setup}}$.

In the following, we define the *computational overhead* of an FSS.

Definition 16 (FSS Cost). *For an $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{FullEval})$ we define the cost functions $C^0_{\mathsf{FSS}}, C^1_{\mathsf{FSS}}$ as the circuit sizes (over the full binary basis) of $\mathsf{FullEval}(0, \cdot)$ and $\mathsf{FullEval}(1, \cdot)$, respectively. If $C^\sigma_{\mathsf{FSS}}(\lambda) = c \cdot N + \mathsf{poly}(\lambda)$ for some constant $c \in O(1)$, for $\sigma \in \{0,1\}$, we say that FSS has* constant overhead.

Multi-point and multi-interval functions. In the following we give the definition of different variants of multi-point functions and multi-interval function. We start by recalling the definition of *t-point function*.

Definition 17 (Multi-point function). *Let $t \in \mathbb{N}, N \in \mathbb{N}$, $\alpha_1, \dots, \alpha_t \in [N]$ pairwise different, $\mathbb{G}$ be an additive group and $\beta_1, \dots, \beta_t \in \mathbb{G}$. Then, we define the t-*point function *specified by $(\alpha_1, \dots, \alpha_t)$ and $\vec{\beta} = (\beta_1, \dots, \beta_t)$ as*

$$f_{\vec{\alpha}}^{\vec{\beta}} \colon [N] \to \mathbb{G},\ f_{\vec{\alpha}}^{\vec{\beta}}(x) := \begin{cases} \beta_i & \text{if } x = \alpha_i \\ 0 & \text{else} \end{cases}.$$

In our constructions we will use a regular version of the multi-point function, where the domain is split into t intervals each of size N/t (assuming $t|N$), such that in each interval exactly one point maps to a non-zero value.

Definition 18 (Regular multi-point function). *Let $t \in \mathbb{N}, N \in \mathbb{N}$, $\alpha_1, \dots, \alpha_t \in [N/t]$, $\mathbb{G}$ be an additive group and $\beta_1, \dots, \beta_t \in \mathbb{G}$. The* regular multi-point function *defined by $\vec{\alpha} = (\alpha_1, \dots, \alpha_t)$ and $\vec{\beta} = (\beta_1, \dots, \beta_t)$ is then*

$$f_{\vec{\alpha}}^{\vec{\beta}} \colon [N] \to \mathbb{G},\ f_{\vec{\alpha}}^{\vec{\beta}}(x) := \begin{cases} \beta_i & \text{if } x = \alpha_i + \frac{N}{t} \cdot (i-1) \\ 0 & \text{else} \end{cases}.$$

Similarly, one can define interval functions. Here, we will start by defining a single interval function. In the literature, this is also referred to as *comparison function*.

Definition 19 (Interval function). *Let $\alpha \in [N]$, $\mathbb{G}$ be an additive group and $\beta \in \mathbb{G}$. Then, we define the* interval function *specified by α and β as*

$$f_{<\alpha}^{\beta} \colon [N] \to \mathbb{G},\ f_{<\alpha}^{\beta}(x) := \begin{cases} \beta & \text{if } x < \alpha \\ 0 & \text{else} \end{cases}.$$

We further consider interval functions with *projected* payload, i.e., where each output $f(x)$ only depends on a subset $\pi_x(\beta)$ of the bits of the payload β.

Definition 20 (Projected-payload interval function). *Let $N \in \mathbb{N}$ be the domain size, $\kappa, \ell \in \mathbb{N}$, $\mathbb{G} = \{0,1\}^\kappa$ be the group of κ-length bit-strings and $\pi_i \colon \{0,1\}^\kappa \to \{0,1\}^\ell$ for $i \in [N]$ projection functions. Let further $\alpha \in [N]$ and $\beta \in \{0,1\}^\kappa$. Then, we define the projected-payload interval function specified by α, β and $\vec{\pi} := (\pi_1, \dots, \pi_N)$ as*

$$f_{<\alpha}^{\beta,\vec{\pi}}(x) = \begin{cases} \pi_x(\beta) & \text{if } x < \alpha \\ 0^\ell & \text{else} \end{cases}.$$

Further, we define the multi-interval version of a projected-payload interval function as follows.

Definition 21 (Projected-payload multi-interval function). *Let $N \in \mathbb{N}$ be the domain size, $\kappa, \ell \in \mathbb{N}$, $\mathbb{G} = \{0,1\}^\kappa$ be the group of κ-length bit-strings and $\pi_i \colon \{0,1\}^\kappa \to \{0,1\}^\ell$ for $i \in [N]$ projection functions. Let further $\alpha_1, \dots, \alpha_t \in [N]$ be pairwise different and $\beta \in \{0,1\}^\kappa$. Then, we define the projected-payload interval function specified by $\vec{\alpha} = (\alpha_1, \dots, \alpha_t)$, β and $\vec{\pi} := (\pi_1, \dots, \pi_N)$ as*

$$f_{<\vec{\alpha}}^{\beta,\vec{\pi}}(x) = \begin{cases} \pi_x(\beta) & \text{if } |\{i \in [t] : \alpha_i < x\}| \equiv 1 \pmod 2 \\ 0^\ell & \text{else} \end{cases}.$$

2.5 Known-Index Function Secret Sharing

In known-index FSS for point functions, introduced in [SGRR19], the index α is allowed to be leaked to party P_0. As observed in [SGRR19, BCG⁺19a], a puncturable PRF suffices to instantiate known-index FSS for point-functions. Similarly, a t-puncturable PRF suffices to instantiate known-index FSS for t-point functions. In [BCG⁺22], it was further observed that allowing to leak the index can also give efficiency improvements for interval FSS, through known-index interval FSS (in their work, this is referred to as *relaxed distributed comparison function*). In the following we give formal definitions of known-index and known-interval FSS.

Definition 22 (Known-index multi-point FSS). *By* known-index multi-point FSS*, we denote an FSS for the class of multi-point functions, with the following two leakage functions:*

- $\mathsf{Leak}_0(f_\alpha^\beta) = (N, \mathbb{G}, t, \alpha)$.
- $\mathsf{Leak}_1(f_\alpha^\beta) = (N, \mathbb{G}, t)$.

In particular, the special index $\alpha \in [N]$ is revealed to party 0.

For $t = 1$, we will also refer to the above as known-index distributed point function (DPF)*.*

Definition 23 (Known-index regular multi-point FSS). *By* known-index regular multi-point FSS*, we denote an FSS for the class of multi-point functions, with the following two leakage functions:*

- $\mathsf{Leak}_0(f_\alpha^\beta) = (N, \mathbb{G}, t, \alpha, \mathsf{reg})$.
- $\mathsf{Leak}_1(f_\alpha^\beta) = (N, \mathbb{G}, t, \mathsf{reg})$.

Here reg *is simply a flag revealing that the noise distribution is regular (rather than arbitrary over $[N]$).*

Definition 24 (Known-index interval FSS/ known-index DCF). *By* known-index interval FSS (or known-index DCF)*, we denote an FSS for the class of interval functions, with the following two leakage functions:*

- $\mathsf{Leak}_0(f_{<\alpha}^\beta) = (N, \mathbb{G}, t, \alpha)$.
- $\mathsf{Leak}_1(f_{<\alpha}^\beta) = (N, \mathbb{G}, t)$.

In particular, the special index $\alpha \in [N]$ is revealed to party 0.

Definition 25 (Known-index projected-payload interval FSS). *By* known-index projected-payload interval FSS*, we denote an FSS for the class of projected-payload interval functions, with the following two leakage functions:*

- $\mathsf{Leak}_0(f_{<\vec{\alpha}}^{\beta,\vec{\pi}}) = (N, \kappa, \ell, \vec{\pi}, \vec{\alpha})$.
- $\mathsf{Leak}_1(f_{<\vec{\alpha}}^{\beta,\vec{\pi}}) = (N, \kappa, \ell, \vec{\pi})$.

In particular, the special index $\alpha \in [N]$ is revealed to party 0.

2.6 Pseudorandom Correlation Generators

We recall the notion of pseudorandom correlation generator (PCG) from [BCG+19b]. At a high level, a PCG for some target ideal correlation takes as input a pair of short, correlated seeds and outputs long correlated pseudorandom strings, where the expansion procedure is deterministic and can be applied locally.

Definition 26 (Correlation Generator). *A PPT algorithm $\mathcal{C}$ is called a* correlation generator*, if $\mathcal{C}$ on input 1^λ outputs a pair of strings in $\{0,1\}^{\ell_0 \cdot N} \times \{0,1\}^{\ell_1 \cdot N}$ for $\ell_0, \ell_1, N \in \mathsf{poly}(\lambda)$.*

The correlation we consider in this paper is the *bit-OT correlation*, where $\ell_0 = \ell_1 = 2$, and $\mathcal{C}$ outputs N uniformly random tuples of the form $((b, s_b), (s_0, s_1))$ (where $b, s_0, s_1 \in \{0,1\}$).

The security definition of PCGs requires the target correlation to satisfy a technical requirement, which roughly says that it is possible to efficiently sample from the conditional distribution of R_0 given $R_1 = r_1$ and vice versa. It is easy to see that this is true for the bit-OT correlation.

Definition 27 (Reverse-sampleable Correlation Generator). *Let $\mathcal{C}$ be a correlation generator. We say $\mathcal{C}$ is* reverse sampleable *if there exists a PPT algorithm* $\mathsf{RSample}$ *such that for $\sigma \in \{0,1\}$ the correlation obtained via:*

$$\{(R_0', R_1') \mid (R_0, R_1) \xleftarrow{\$} \mathcal{C}(1^\lambda), R_\sigma' := R_\sigma, R_{1-\sigma}' \xleftarrow{\$} \mathsf{RSample}(\sigma, R_\sigma)\}$$

is computationally indistinguishable from $\mathcal{C}(1^\lambda)$.

The following definition of pseudorandom correlation generators is taken almost verbatim from [BCG+19b]; it generalizes an earlier definition of pseudorandom VOLE generator in [BCGI18].

Definition 28 (Pseudorandom Correlation Generator (PCG) [BCG+19b]). *Let $\mathcal{C}$ be a reverse-sampleable correlation generator. A* PCG for $\mathcal{C}$ *is a pair of algorithms* $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *with the following syntax:*

– $\mathsf{PCG.Gen}(1^\lambda)$ *is a PPT algorithm that given a security parameter* λ*, outputs a pair of seeds* $(\mathsf{k}_0, \mathsf{k}_1)$*;*
– $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ *is a polynomial-time algorithm that given party index* $\sigma \in \{0,1\}$ *and a seed* k_σ*, outputs a bit string* $R_\sigma \in \{0,1\}^{\ell_\sigma}$*.*

The algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *should satisfy the following:*

– **Correctness.** *The correlation obtained via:*

$$\{(R_0, R_1) \,|(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), (R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma))_{\sigma=0,1}\}$$

is computationally indistinguishable from $\mathcal{C}(1^\lambda)$*.*
– **Security.** *For any* $\sigma \in \{0,1\}$*, the following two distributions are computationally indistinguishable:*

$$\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)\} \textit{ and}$$
$$\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_{1-\sigma} \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_{1-\sigma}),$$
$$R_\sigma \xleftarrow{\$} \mathsf{RSample}(\sigma, R_{1-\sigma})\}$$

where $\mathsf{RSample}$ *is the reverse sampling algorithm for correlation* $\mathcal{C}$*.*

3 Low-Complexity Known-Index Function Secret Sharing

We now give details on the constructions of known-index FSS we use, and analyze their circuit complexity. First, we recall definitions such as point and interval functions. Next, we detail constructions for known-index DPF and DCF, which are based on prior works. Then, in Section 3.3, we give our new construction of projected-payload interval FSS, which uses DPF and DCF as a black-box.

3.1 Known-Index Distributed Point Function

We use two constructions of known-index DPF, based on standard techniques. Firstly, a naive one with $O(\sqrt{N})$ size setup, and secondly, a tree-based construction with $O(\log N)$ setup. The first construction, however, has roughly half the circuit complexity for $\mathsf{FullEval}$.

Square-root construction. In this construction, the Setup algorithm samples $\sqrt{N}$ PRG keys, each of which is later expanded to produce $\sqrt{N}$ pseudorandom outputs. All these keys are given to party 1, while party 0 is missing one key (depending on α). Party 1 additionally gets a vector of bits, indicating which key is missing, and finally a correction word CW to fix the missing output to be the right value.

The construction is shown in Fig. 1, where the optimal parameters set $n_0 = n_1 = \sqrt{N}$. The parameter ℓ is the output length of the point function, in bits. Note that the complexity of $\mathsf{FullEval}$ when $\sigma = 1$ is exactly the cost of generating $N\ell$ pseudorandom bits using G. When $\sigma = 0$, there is an additional cost of $2N\ell$ gates. This gives amortized costs of:

– $C^0_{\mathsf{FSS}}(\lambda) = C_{\mathsf{prg}}(\ell) + 2\ell$
– $C^1_{\mathsf{FSS}}(\lambda) = C_{\mathsf{prg}}(\ell)$

The Setup algorithm, meanwhile, has a cost in $O(n_0\lambda + C_{\mathsf{prg}}(\ell n_1))$.

Logarithmic construction. The standard construction for known-index DPF is based on a puncturable PRF, as used in [BCG+19a, SGRR19], and shown in Fig. 2. We consider a point function of domain $[N]$ and range $\{0,1\}^\lambda$. We later discuss how to modify the construction for smaller ranges. We let $\alpha \in [N]$ be the hidden index and $(\alpha_{\log N}, \ldots, \alpha_1)$ its bit decomposition, so $\alpha = \sum_{i=0}^{\log N - 1} 2^i \cdot \alpha_{\log N - i}$.

The construction uses a PRG key k to define a complete binary tree with N leaves, where the key k is assigned to the root node, and at each non-leaf node, the two children are obtained by applying the PRG to the key of the parent. The FSS key of party 0, who learns the point α, is obtaining by giving out one key for each level of the tree, which allows reconstructing all-but-one of the leaves. A final correction word CW is used to fill in the missing value to be a share of β.

```
Construction: Square-root known-index DPF

Building Blocks: Let $N = n_0 n_1$, and $G : \{0,1\}^\lambda \to \{0,1\}^{\ell n_1}$ be a PRG.

$\mathsf{Setup}(1^\lambda, N, \alpha, \beta)$:

  1. Write $\alpha = \alpha_0 + \alpha_1 n_1 \bmod N$ for $\alpha_0 \in \{0,\dots,n_0-1\}, \alpha_1 \in \{0,\dots,n_1-1\}$
  2. Sample $k_0,\dots,k_{n_0-1} \leftarrow \{0,1\}^\lambda$.
  3. Compute $CW = G(k_{\alpha_0}) \oplus (0,\dots,\beta,\dots,0)$, where the latter vector is 0 everywhere except positions
     $\{\alpha_1,\dots,\alpha_1+\ell-1\}$.
  4. Let $(k'_1,\dots,k'_{n_0})$ equal $(k_1,\dots,k_{n_0})$, except $k_{\alpha_0} = 0^\lambda$
  5. Let $(c_0,\dots,c_{n_0-1}) = 0^{n_0}$, except $c_{\alpha_0} = 1$
  6. Output $\mathsf{k}_0 = (CW, \{c_i, k'_i\}_{i\in[n_0]})$ and $\mathsf{k}_1 = \{k_i\}_{i\in[n_0]}$

$\mathsf{FullEval}(\sigma, \mathsf{k}_\sigma)$:

  1. If $\sigma = 0$:
     (a) Parse $\mathsf{k}_0 = (\alpha, CW, \{c_i, k'_i\}_{i\in[n_0]})$
     (b) For $i \in \{0,\dots,n_0-1\}$, let $(v_{in_1+1},\dots,v_{in_1+n_1}) = G(k'_i) \oplus c_i \cdot CW$
     (c) Output $(v_1,\dots,v_N)$
  2. If $\sigma = 1$:
     (a) For $i \in \{0,\dots,n_0-1\}$, let $(v_{in_1+1},\dots,v_{in_1+n_1}) = G(k_i)$
     (b) Output $(v_1,\dots,v_N)$
```

Fig. 1: Known-index DPF with square-root complexity, for domain $[N]$ and range $\{0,1\}^\ell$

Circuit Complexity. For the setup phase, if the PRG has constant overhead then the complexity is $O(\log N\lambda)$. For $\mathsf{FullEval}$, the simplest case is when $\sigma = 1$. Here, the cost of $\mathsf{FullEval}$ is simply $N-1$ evaluations of G, which costs $C_{\mathsf{prg}}(2\lambda(N-1))$ gates. We then have an amortized cost

$$C^1_{\mathsf{kiDPF}}(\lambda) \le C_{\mathsf{prg}}(2\lambda)$$

If $\sigma = 0$, we define a circuit as follows. First, expand each key $\overline{K}^i$ into its $2^{\log N - i}$ children of the tree, for $i = 1,\dots\log N - 1$. This costs $(N/2-1)+(N/4-1)+\cdots+(2-1)+(1-1) = N - \log N - 1$ evaluations of G. At this point, we have the correct $N-1$ outputs of $\mathsf{FullEval}$, but in the wrong order (and with the “missing” value in the last position). We can permute these values into the right order using the bits α_i, as follows.

For $j = 0,\dots,\log N - 1$:

1. Let $i = \log N - j$
2. If $\alpha_i = 1$, swap the values in positions $(N,\dots,N-2^j+1)$ and $(N-2^j,\dots,N-2^j-2^j+1)$

A conditional swap of a pair of bits (that is, a MUX gate) can be done with 4 Boolean gates. Since the outputs are each of length λ bits, the above permutation costs $\lambda\cdot(1+2+4+\cdots+N/2) = \lambda\cdot(N-1)$ swaps of bits. Combining with the PRG evaluations, we get an overall cost for $\mathsf{FullEval}$ of

$$C_{\mathsf{prg}}(2\lambda(N-\log N-1)) + 4\lambda(N-1)$$

for party 0, giving an amortized cost of $C^0_{\mathsf{kiDPF}}(\lambda) \le C_{\mathsf{prg}}(2\lambda) + 4\lambda$.

Complexity for Smaller Output Lengths. When the output length is $\ell < \lambda$ bits, we optimize the above construction by truncating the lower $\log(\lambda/\ell)$ levels of the tree. Each λ-bit leaf in the truncated tree is then used to define λ/ℓ output values.

The cost of $\mathsf{FullEval}$ then becomes the same as that of $\mathsf{FullEval}$ on a domain of size $N\ell/\lambda$, with λ-bit outputs. That gives:

- $C^0_{\mathsf{FSS}}(\lambda) \le C_{\mathsf{prg}}(2\ell) + 4\ell$
- $C^1_{\mathsf{FSS}}(\lambda) \le C_{\mathsf{prg}}(2\ell)$

Construction: Known-index DPF

```
BUILDING BLOCKS: PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$

$\mathsf{Setup}(1^\lambda, N, \alpha, \beta)$:

  1. Sample $k \leftarrow \{0,1\}^\lambda$
  2. Let $K = k$
  3. For $i \in \{1, \ldots, \log N\}$:
     (a) Compute $(K^i_0, K^i_1) = G(K)$
     (b) Let $\overline{K}^i = K^i_{1-\alpha_i}$ and update $K \leftarrow K^i_{\alpha_i}$
  4. Compute $CW = K \oplus \beta$.
  5. Output $\mathsf{k}_0 = (CW, \{\alpha_i, \overline{K}^i\}_i)$ and $\mathsf{k}_1 = k$.

$\mathsf{FullEval}(\sigma, \mathsf{k}_\sigma)$:

  1. If $\sigma = 0$:
     (a) Parse $\mathsf{k}_1 = (CW, \{\alpha_i, \overline{K}^i\}_i)$
     (b) Set $K^1_{1-\alpha_1} = \overline{K}^1$
     (c) For $i \in \{2, \ldots, \log N\}$:
         – Let $x = \sum_{j=1}^{i-1} 2^{j-1} \cdot \alpha_{i-j}$
         – For $j \in \{0, \ldots, 2^{i-1} - 1\} \setminus \{x\}$, compute $(K^i_{2j}, K^i_{2j+1}) = G(K^{i-1}_j)$
         – Let $K^i_{2x+\overline{\alpha_i}} = \overline{K_i}$
     (d) Let $K^{\log N}_\alpha = CW$
     (e) Output $(v_1, \ldots, v_N) = (K^{\log N}_0, \ldots, K^{\log N}_{N-1})$
  2. If $\sigma = 1$:
     (a) Let $K^0_0 = \mathsf{k}_1$
     (b) For $i = \{1, \ldots, \log N\}$:
         – For $j \in \{0, \ldots, 2^{i-1} - 1\}$, compute $(K^i_{2j}, K^i_{2j+1}) = G(K^{i-1}_j)$
     (c) Output $(v_1, \ldots, v_n) = (K^{\log N}_0, \ldots, K^{\log N}_{N-1})$
```

Fig. 2: Known-index DPF based on a GGM-style puncturable PRF

3.2 Known-Index Interval FSS

We use the construction from [BCG+21], which is shown in Fig. 3, rephrased in a way so it is easier to analyze the circuit complexity. The construction works over a general group $(\mathbb{G}, +)$, which in our setting will be $(\{0,1\}^\kappa, \oplus)$.

Complexity of Setup. By inspection, the Setup phase has cost $O(\log N(\lambda + \kappa)$. This scales only logarithmically with N, so satisfies the succinctness requirement.

Complexity of FullEval. We first consider $\sigma = 1$. There are $N - 1$ calls to the PRG G, $N - 1$ calls to C, and $2N - 1$ XORs of κ-bit strings. This gives an amortized cost per output of

$$C^1_{\mathsf{kiDCF}}(\lambda) = C_{\mathsf{prg}}(2\lambda + \kappa) + 2\kappa$$

For $\sigma = 0$, we construct circuit in a similar way to the tree-based known-index DPF construction. The circuit will first compute all the pairs $(K^{\log N}_j, \gamma^{\log N}_i)$, for $j \neq \alpha$, by expanding G and C on the $\overline{K}^i$ values to obtain all their descendants. The resulting $N-1$ pairs are ordered such that the first $N/2$ come from $\overline{K}^1$, the next $N/4$ from $\overline{K}^2$ etc. The value y is placed into the N-th position of the list. Then, the values are permuted by first swapping the last two tuples, labelled $N - 1$ and N, depending on $\alpha_{\log N}$, then swapping tuples $\{N-3, N-2\}$ and $\{N-1, N\}$ depending on $\alpha_{\log N-1}$, and so on. Since each tuple has size $\leq \lambda + \kappa$ bits, there are $(\lambda + \kappa)(N - 1)$ swaps of bits in all.

The cost of the first phase of this is $N - \log N - 1$ evaluations of G and C, plus $N - \log N - 1$ XORs of κ-bit strings. Combined with the cost of 4 gates for each swap of a pair of bits, we get an amortized cost of

$$C^0_{\mathsf{kiDCF}}(\lambda) \leq C_{\mathsf{prg}}(2\lambda + \kappa) + 5\kappa + 4\lambda$$

Construction: Known-index Interval FSS

Building Blocks: Let $(\mathbb{G}, +)$ be an abelian group and $\mathsf{Convert}_{\mathbb{G}} : \{0,1\}^\lambda \to \mathbb{G}$ be a map converting a random λ-bit string to a pseudorandom group element in $\mathbb{G}$.
Functions $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ and $C : \{0,1\}^\lambda \to \mathbb{G}$ such that $G(k) := C(k) \| G_0(k) \| G_1(k)$ is a secure PRG.

$\mathsf{Setup}(1^\lambda, N, \alpha, \beta)$:

```
1. Let $(\alpha_{\log N}, \dots, \alpha_1)$ be such that $\alpha = \sum_{i=0}^{\log N - 1} 2^i \cdot \alpha_{\log N - i}$
2. Sample $k \leftarrow \{0,1\}^\lambda$ and let $K \leftarrow k$.
3. Let $C^0 = \overline{C}^0 = 0 \in \mathbb{G}$
4. For $i \in [\log N]$, compute the following:
   (a) $(K_0^i, K_1^i) = G(K)$ and $\gamma_i = C(K)$
   (b) $\overline{K}^i = K^i_{\overline{\alpha_i}}$
   (c) $K \leftarrow K^i_{\alpha_i}$
   (d) $(c_i, \bar{c}_i) = \begin{cases} (\gamma_i, 0) & \text{if } \alpha_i = 0, \\ (0, \gamma_i) & \text{else} \end{cases}$
   (e) $\overline{S}^i = \bar{c}_i + C^{i-1}$
   (f) $\overline{B}^i = \overline{S}_i + \alpha_i \cdot \beta$
   (g) $C^i = C^{i-1} + c_i$
5. Define $y = \mathsf{Convert}_{\mathbb{G}}(K) + C^{\log N}$
6. Set $\mathsf{k}_0 = (y, \{\alpha_i, \overline{K}^i, \overline{B}^i\}_{i \in [\log N]})$ and $\mathsf{k}_1 = k$.
7. Output $(\mathsf{k}_0, \mathsf{k}_1)$.
```

$\mathsf{FullEval}(\sigma, \mathsf{k}_\sigma)$:

```
1. If $\sigma = 0$:
   (a) Parse $\mathsf{k}_0 = (y, \{\alpha_i, \overline{K}^i, \overline{B}^i\}_{i \in [\log N]})$
   (b) Set $K^1_{\overline{\alpha_1}} = \overline{K}^1$
   (c) For $i \in \{2, \dots, \log N\}$:
       – Let $x = \sum_{j=1}^{i-1} 2^{j-1} \cdot \alpha_{i-j}$
       – For $j \in \{0, \dots, 2^{i-1} - 1\} \setminus \{x\}$:
           • Compute $(K^i_{2j}, K^i_{2j+1}) = G(K^{i-1}_j)$
           • Compute $\gamma^i_{2j} = \gamma^{i-1}_j + C(K^{i-1}_j)$ and $\gamma^i_{2j+1} = \gamma^{i-1}_j$.
       – Let $K^i_{2x+\overline{\alpha_i}} = \overline{K_i}$
       – Let $\gamma^i_{2x+\overline{\alpha_i}} = \overline{B}^i$
   (d) Let $v_\alpha = y$
   (e) Compute $v_i = \mathsf{Convert}_{\mathbb{G}}(K_i^{\log N}) + \gamma_i^{\log N}$, for $i \in [N] \setminus \{\alpha\}$
   (f) Output $(v_1, \dots, v_N)$
2. If $\sigma = 1$:
   (a) Let $K^0_0 = \mathsf{k}_1$ and $\gamma^0_0 = 0 \in \mathbb{G}$
   (b) For $i \in \{1, \dots, \log N\}$:
       – For $j \in \{0, \dots, 2^{i-1} - 1\}$:
           • Compute $(K^i_{2j}, K^i_{2j+1}) = G(K^{i-1}_j)$
           • Compute $\gamma^i_{2j} = \gamma^{i-1}_j + C(K^{i-1}_j)$ and $\gamma^i_{2j+1} = \gamma^{i-1}_j$.
   (c) Compute $v_i = \mathsf{Convert}_{\mathbb{G}}(K_i^{\log N}) + \gamma_i^{\log N}$, for $i \in [N]$
   (d) Output $(v_1, \dots, v_n)$
```

Fig. 3: Known-index interval FSS

3.3 Projected-Payload Interval FSS

Our construction of a known-index projected-payload interval FSS can be found in Figure 4. The main idea is to reduce the problem to a standard (known-index) interval FSS (or DCF) on a smaller domain of size N/κ, where κ is the payload length.

First, note that with a single DCF of this kind, using the index $\alpha' = \lceil \alpha/\kappa \rceil$ and payload β, the parties can already obtain shares of the projected-payload function evaluated at all $i \in [N]$ except for a subset of size κ, corresponding to indices i where $\lceil i/\kappa \rceil = \alpha'$. At this subset, the DCF will instead always give shares of zero.

We then combine this with a DPF to allow correcting the values where $\lceil i/\kappa \rceil = \alpha'$ to the right value. Concretely, we use a DPF with index α' and random payload r, and have both parties always add the DPF evaluation (expanded to κ bits with a PRG) to the DCF one. Then, party 0, who knows α', is given a correction word CW, which is only added to the evaluations i where $\lceil i/\kappa \rceil = \alpha'$. CW is computed in the setup phase such that it corrects these outputs to be shares of the projected-payload function, as required.

Theorem 29. *The construction in Fig. 4 is a secure known-index, projected-payload interval FSS scheme.*

Proof. We first show correctness and then security, assuming that DPF and DCF are both correct and secure.

Correctness. We show that, for any pair of keys $(\mathsf{k}_0, \mathsf{k}_1)$ output by Setup, it holds that

$$\mathsf{FullEval}(0, \mathsf{k}_0) \oplus \mathsf{FullEval}(1, \mathsf{k}_1) = \left(f_{<\alpha}^{\beta,\vec{\pi}}(1), \ldots, f_{<\alpha}^{\beta,\vec{\pi}}(N) \right)$$

except with negligible probability. Recall that

$$f_{<\alpha}^{\beta,\vec{\pi}}(i) = \begin{cases} \pi_i(\beta) & \text{if } i < \alpha \\ 0^\ell & \text{else} \end{cases}$$

Let $(v_{0,1}, \ldots, v_{0,N})$ and $(v_{1,1}, \ldots, v_{1,N})$ be the two outputs of FullEval, and $y_{\sigma_i}, z_{\sigma_i}$ be the intermediate values computed in FullEval. Write $i = (i' - 1)\kappa + j$ for $j \in \{0, \ldots, \kappa - 1\}$ and $i' = \lceil i/N \rceil$, and view j as an element of $[n]$ by mapping 0 to κ. Consider first the case where $i' \neq \alpha'$. Then, party 0 never uses the correction word, and furthermore we have $y_{0,i'} \oplus y_{1,i'} = 0$, by the correctness of DPF. We then have

$$\begin{aligned} v_{0,i} \oplus v_{1,i} &= G(y_{0,i'})_j \oplus \pi_i(z_{0,i'}) \oplus G(y_{1,i'})_j \oplus \pi_i(z_{1,i'}) \\ &= \pi_i(z_{0,i'} \oplus z_{1,i'}) \\ &= \pi_i(f_{<\alpha'}^{\beta}(i')) \\ &= f_{<\alpha}^{\beta}(i) \end{aligned}$$

where the penultimate line applies the correctness of DCF, and the final line holds because $i' \neq \alpha'$.

For the κ values of i where $\lceil \alpha/N \rceil = \alpha'$, party 0's outputs are $CW_1, \ldots, CW_\kappa$, where $CW_j = u_j \oplus \pi_j(z_{1,\alpha'}) \oplus f_{<\alpha}^{\beta,\vec{\pi}}(i)$. Again writing $i = (\alpha' - 1)\kappa + j$, we have

$$\begin{aligned} v_{0,i} \oplus v_{1,i} &= u_j \oplus \pi_i(z_{1,\alpha'}) \oplus f_{<\alpha}^{\beta,\vec{\pi}}(i) \oplus G(y_{1,\alpha'})_j \oplus \pi_i(z_{1,\alpha'}) \\ &= f_{<\alpha}^{\beta,\vec{\pi}}(i) \end{aligned}$$

as required.

Security. Suppose first that party 0 is corrupted. We show that for any α, β, β', the distribution of k_0 generated for the function $f_{<\alpha}^{\beta,\vec{\pi}}$ is indistinguishable from the one for $f_{<\alpha}^{\beta',\vec{\pi}}$. This then implies the simulation-based security definition, as noted in [BGI16]. The key k_0 consists of α, the correction words CW and the two FSS keys $k_0^{\mathsf{DPF}}, k_0^{\mathsf{DCF}}$.

Game 1. We start by computing $y_{1,\alpha'}$ using key k_0^{DPF}, instead of k_1^{DPF}, namely, $y_{1,\alpha'} = \mathsf{DPF.Eval}(k_1^{\mathsf{DPF}}, \alpha') \oplus r$. This is indistinguishable from the original distribution, due to the correctness of DPF.

Game 2. Next, the key k_0^{DPF} is replaced with one corresponding to a random payload r' instead of r. This is indistinguishable from **Game 1**, due to the security of DPF.

Game 3. Next, instead of computing u from the PRG G, we sample it uniformly. Note that CW is then uniformly random. This hybrid is indistinguishable thanks to the security of G, because the PRG seed is masked with r, which is independent of all other values.

Construction: Known-index projected-payload interval FSS

BUILDING BLOCKS:

- Known-index single-point FSS ($\mathsf{DPF.Setup}, \mathsf{DPF.Eval}$)
- Known index single-interval FSS ($\mathsf{DCF.Setup}, \mathsf{DCF.Eval}$)
- PRG $G : \{0,1\}^\lambda \to \{0,1\}^{\ell\cdot\kappa}$

$\mathsf{Setup}(1^\lambda, f^{\beta,\vec{\pi}}_{<\alpha})$:

```
1. Parse description of $f^{\beta,\vec{\pi}}_{<\alpha}$ to obtain the domain size $N$, output length $\kappa$, projected output length $\ell$.
2. Let $\alpha' = \lceil \alpha/\kappa \rceil$.
3. Generate DPF for $N/\kappa$ $\lambda$-bit PRG seeds, with one seed punctured:
   (a) Sample random $r \leftarrow \{0,1\}^\lambda$. (To hide punctured key)
   (b) Let $f^r_{\alpha'}$ be the point function with input domain $[N/\kappa]$ and range $\{0,1\}^\lambda$.
   (c) Run $(k_0^{\mathsf{DPF}}, k_1^{\mathsf{DPF}}) \leftarrow \mathsf{DPF.Setup}(1^\lambda, f^r_{\alpha'})$
4. Generate DCF for $N/\kappa$ $\kappa$-bit "output corrections", corresponding to $\beta \in \{0,1\}^\kappa$ for $i < \alpha'$ and $0^\kappa$ for $i \geq \alpha'$
   (a) Let $f^\beta_{<\alpha'}$ be the comparison function with input domain $[N/\kappa]$ and range $\{0,1\}^\kappa$.
   (b) Run $(k_0^{\mathsf{DCF}}, k_1^{\mathsf{DCF}}) \leftarrow \mathsf{DCF.Setup}(1^\lambda, f^\beta_{<\alpha'})$.
5. Generate $\kappa$-bit punctured correction value (for prefix $\alpha'$)
   (a) Evaluate party 1's keys at $\alpha'$, to get $y_{1,\alpha'} = \mathsf{DPF.Eval}(k_1^{\mathsf{DPF}}, \alpha')$ and $z_{1,\alpha'} = \mathsf{DCF.Eval}(k_1^{\mathsf{DCF}}, \alpha') \in \{0,1\}^\kappa$.
   (b) Compute $u = G(y_{1,\alpha'})$. Parse as $u = (u_1, \dots, u_\kappa) \in (\{0,1\}^\ell)^\kappa$.
   (c) For each $i \in [\kappa]$:
       – Let $j = (\alpha' - 1) \cdot \kappa + i$
       – Compute masked output $CW_i = u_i \oplus \pi_i(z_{i,\alpha',j}) \oplus f^{\beta,\vec{\pi}}_{<\alpha}(j) \in \{0,1\}^\ell$.
   (d) Let $CW = (CW_1, \dots, CW_\kappa)$.
6. Output $((k_0^{\mathsf{DPF}}, k_0^{\mathsf{DCF}}, CW, \alpha), (k_1^{\mathsf{DPF}}, k_1^{\mathsf{DCF}}))$.
```

$\mathsf{FullEval}(\sigma, k_\sigma)$:

```
1. Parse key as $k_0 = (k_0^{\mathsf{DPF}}, k_0^{\mathsf{DCF}}, CW, \alpha)$ or $k_1 = (k_1^{\mathsf{DPF}}, k_1^{\mathsf{DCF}})$
2. Compute $(y_{\sigma,1}, \dots, y_{\sigma,N/\kappa}) = \mathsf{DPF.FullEval}(\sigma, k_\sigma^{\mathsf{DPF}})$ in $(\{0,1\}^\lambda)^{N/\kappa}$.
3. Compute $(z_{\sigma,1}, \dots, z_{\sigma,N/\kappa}) = \mathsf{DCF.FullEval}(\sigma, k_\sigma^{\mathsf{DCF}})$ in $(\{0,1\}^\kappa)^{N/\kappa}$.
4. For $i = 1, \dots, N$:
   (a) Let $i' = \lceil i/N \rceil$ and $j = i \mod \kappa$, mapped into $[\kappa]$
   (b) Let $v_{\sigma,i} = G(y_{\sigma,i'})_j \oplus \pi_i(z_{\sigma,i'})$.
5. If $\sigma = 0$:
   (a) Let $\alpha' = \lceil \alpha/\kappa \rceil$.
   (b) Let $(v_{0,(\alpha'-1)\cdot\kappa+1}, \dots, v_{0,(\alpha'-1)\cdot\kappa+\kappa}) := CW$.
6. Output $(v_{\sigma,1}, \dots, v_{\sigma,N})$.
```

Fig. 4: Known-index projected-payload interval FSS

Game 4. In this hybrid, we replace the DCF key k_0^{DCF} with one using the payload β'. **Game 4** is indistinguishable from **Game 1**, by the security of DCF.

Games 5 onwards. Here, we reverse the steps of the previous hybrids, except using β' in place of β to generate the correction words. The final hybrid is the same as the one we started with, except using β' instead of β, which completes the argument.

The case when party 1 is corrupted is straightforward, since key k_1 only contains a DPF key and an DCF key. Security in this case is then immediate, from the underlying security of these two primitives. □

Complexity. The size of each party's key is exactly that of a DPF key and DCF key of domain size N/κ and respective ranges $\{0,1\}^\lambda$, $\{0,1\}^\kappa$. Party 0 additionally has $\kappa\ell + \log N$ bits for the correction word and α. The asymptotic complexity of the setup algorithm is that of the setup for DPF and DCF, plus $O(\ell\kappa) = O(\kappa)$, assuming ℓ is constant and the PRG has constant overhead. The DPF setup costs $O(\log(N/\kappa)\lambda$, while the DCF setup costs $O(\log(N/\kappa)(\lambda+\kappa)$, so the overall cost is dominated by $O(\log(N/\kappa)(\lambda+\kappa)$.

In FullEval, both parties compute one FullEval for each of DPF and DCF, plus generate $N\ell$ pseudo-random bits using G, and $N\ell$ XORs to add the projected z bits. Party 0 then replaces $\ell\kappa$ of these bits with CW, depending on α'. If α' is encoded in k_0 as a one-hot vector, this can be done with $2N\ell$ extra operations. We therefore get costs of

$$C^0_{\mathsf{FSS}}(\lambda) = C_{\mathsf{DPF}}(\lambda) + C_{\mathsf{DCF}}(\lambda) + N\ell + (1-\sigma)2N\ell$$

3.4 Extension to Multi-Point/Multi-Interval Functions.

All the constructions we present in this section are described for single-point functions, or single-interval functions. They can all easily be extended to multi-point functions for t points $(\alpha_1,\beta_1),\ldots,(\alpha_t,\beta_t)$, or multi-interval functions, as used in our main constructions. If the indices α_i are known to have a *regular* pattern (as in LPN with regular noise), we can simply concatenate the outputs of t independent instances of single-point DPF, with a domain size of N/t. The resulting complexity is t times that of the N/t-point DPF.

4 Constant-Overhead PCG for OT from Primal LPN

In this section we give a PCG for OT with constant overhead in Figure 5. An inherent limitation to this approach is that primal LPN is limited to subquadratic stretch.

First, following Alekhnovich [Ale03], we will consider a primal code generation procedure that outputs matrices $\mathbf{G}$ that are *very sparse*. In particular, $\mathbf{G}$ will be sampled uniformly at random subject to the constraint that every row has exactly d 1's. Alekhnovich already conjectured this is hard when $d = 3$ if $N, t = O(n)$, where the noise is sampled to have weight t. Polynomial-time attacks exist with $N = \Omega(n^{d/2})$ [App16, BSV19]: one hopes for there to be two rows of $\mathbf{G}$ which agree (which occurs with probability $\frac{\binom{N}{2}}{\binom{n}{d}}$). This is the same as saying the dual distance of $\mathbf{G}$ is 2.

However, as discussed in Section 2.3 when the dual distance D is larger we obtain security against linear tests: we achieve security $2^{-\lambda}$ when $Dt \geq (\ln 2)\lambda N/2$. In general, for any $\gamma > 0$ it is feasible to have a d-sparse matrix $\mathbf{G} \in \{0,1\}^{N\times n}$ with dual distance $D = \Omega_d(n^\gamma)$ and $N = n^{\frac{1-\gamma}{2}d+\gamma}$. In particular we can choose $\gamma = 9/10$ to get $D = \Theta_d(n^{9/10})$ and $N = n^{\frac{d+18}{20}}$, so if we wish to have $N\lambda = O(tD)$ to guarantee exponentially small in λ security against linear tests we may choose $t = \Theta_d(\lambda n^{\frac{d}{18+d}})$.

We must also be careful in light of the attack by Arora and Ge [AG11], which is effective when $N = \Omega(n^2)$. For this reason, we will ensure $N = o(n^2)$.

In what follows (and in the rest of the paper), we assume the existence of an *explicitly generated* matrix $\mathbf{G}$ with sparsity $d = O(1)$ for which the primal $(\mathsf{Reg}_t^N(\{0,1\}), \mathbf{G}, \{0,1\})$-$\mathsf{LPN}(n,N)$ holds with $n, t \leq N^{1-\gamma}$ for some $\gamma > 0$. Alternatively, we conjecture that the randomized expander generation algorithm from [AK19] can be used to efficiently generate such $\mathbf{G}$ with negligible failure probability.

We show security of the PCG in the theorem below, and then analyze its overhead.

Theorem 30. *Let $N = N(\lambda), n = n(\lambda), t = t(\lambda), \kappa = \kappa(\lambda) \in \mathbb{N}$, let $\ell, d \in \mathbb{N}$ be constants and let $\mathbf{C}$ a primal code generation algorithm with sparsity d (i.e., generating code matrices where each row has at most d non-zero entries). If the (primal) $(\mathsf{Reg}_t^N(\mathbb{F}_2), \mathbf{C}, \mathbb{F}_2)$-$\mathsf{LPN}(n,N)$-assumption holds, if $G\colon \{0,1\}^\kappa \to \{0,1\}^N$ is a correlation-robust ℓ-local PRG, and if $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{FullEval})$ is a known-index regular t-point FSS, then the PCG as defined in Figure 5 is a PCG for generating N instances of the bit-OT correlation.*

Construction $\mathsf{PCG}_{\mathsf{OT}}^{\mathsf{primal}}$

PARAMETERS:

- Security parameter $\lambda \in \mathbb{N}$, matrix parameters $N = N(\lambda), n = n(\lambda) \in \mathbb{N}$ with $N > n$, constant matrix sparsity parameter $d \in \mathbb{N}$, noise weight $t = t(\lambda) \in \mathbb{N}$, local PRG input length $\kappa = \kappa(\lambda) \in \mathbb{N}$, constant locality $\ell \in \mathbb{N}$.
- A *primal* sparse code generation algorithm $\mathbf{C}$ returning matrices in $\{0,1\}^{N \times n}$ with d non-zero entries per row and a public matrix $\mathbf{G} \overset{\$}{\leftarrow} \mathbf{C}(N, n, \mathbb{F}_2)$ sampled according to $\mathbf{C}$.
- A constant-overhead known-index regular t-point FSS $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{FullEval})$ over domain $[N]$ and range $\{0,1\}^\ell$.
- A correlation-robust ℓ-local PRG $G\colon \{0,1\}^\kappa \to \{0,1\}^N$ with $G(x) = P_1(\pi_1(x)) \| \ldots \| P_N(\pi_N(x))$ for all $x \in \{0,1\}^\kappa$.

CORRELATION: Outputs N tuples $((b, w), (w_0, w_1))$, where b, w_0, w_1 are random bits and $w = w_b$.

GEN:

- Pick a *local PRG seed* $x \overset{\$}{\leftarrow} \{0,1\}^\kappa$ at random.
- Pick an *LPN seed* $\vec{s} \overset{\$}{\leftarrow} \{0,1\}^n$ at random.
- Generate a random additive secret sharing of $\vec{r} := (s_1 \cdot x, s_2 \cdot x, \ldots, s_N \cdot x)$, i.e., choose $\vec{r}_1 \overset{\$}{\leftarrow} (\{0,1\}^\kappa)^n$ and set $\vec{r}_1 := \vec{r}_0 \oplus \vec{r}$.
- Choose *regular noise positions* $\vec{\alpha} \overset{\$}{\leftarrow} [N/t]^t$ at random.
- Set $\beta_i := \pi_{\alpha_i + \frac{N}{t} \cdot (i-1)}(x) \in \{0,1\}^\ell$ for each $i \in [t]$ and set $\vec{\beta} := (\beta_1, \ldots, \beta_t)$.
- Set $(K_0, K_1) \leftarrow \mathsf{FSS.Setup}(1^\lambda, \vec{\alpha}, \vec{\beta})$.
- Set $\mathsf{k}_0 := (\vec{s}, \vec{r}_0, \vec{\alpha}, K_0)$ and $\mathsf{k}_1 := (x, \vec{r}_1, K_1)$ and output $(\mathsf{k}_0, \mathsf{k}_1)$.

EXPAND: On input $(\sigma, \mathsf{k}_\sigma)$:

1. If $\sigma = 0$, parse k_0 as $(\vec{s}, \vec{r}_0, \vec{\alpha}, K_0)$ and proceed as follows:
   - Let $\vec{\mu} \in \{0,1\}^N$ be the regular noise vector defined by $\vec{\alpha}$, i.e.,
   $$\mu_j = \begin{cases} 1 & \text{if } j = \alpha_i + \frac{N}{t} \cdot (i-1) \\ 0 & \text{else} \end{cases}.$$
   - Set $\vec{b} := \mathbf{G} \cdot \vec{s} \oplus \vec{\mu} \in \{0,1\}^N$.
   - Set $\vec{y}^0 := \mathbf{G} \cdot \vec{r}_0 \in (\{0,1\}^\kappa)^N$. //Note that we only need the ℓ entries $\pi_j(\vec{y}^0_j) \in \{0,1\}^\ell$ to continue. Towards constant overhead this step can therefore be computed in $N \cdot d \cdot \ell \in O(N)$ operations (by only computing relevant parts of the matrix-vector product).
   - Compute $\vec{v}^0 \leftarrow \mathsf{FSS.FullEval}(0, K_0) \in (\{0,1\}^\ell)^N$.
   - For each $j \in [N]$, compute $w_j := P_j(\pi_j(y^0_j) \oplus v^0_j) \in \{0,1\}$.
   - Output $\{(b_j, w_j)\}_{j \in [N]}$.
2. If $\sigma = 1$, parse k_1 as $(\vec{x}, \vec{r}_1, K_1)$ and proceed as follows:
   - Set $\vec{y}^1 := \mathbf{G} \cdot \vec{r}_1 \in (\{0,1\}^\kappa)^N$.
   - Compute $\vec{v}_1 \leftarrow \mathsf{FSS.FullEval}(1, K_1) \in (\{0,1\}^\ell)^N$.
   - For $j \in [N]$, $b \in \{0,1\}$, compute $w_{j,b} := P_j(\pi_j(y^1_j) \oplus \vec{v}^1_j \oplus b \cdot \pi_j(\vec{x}))$.
   - Output $\{(w_{j,0}, w_{j,1})\}_{j \in [N]}$.

Fig. 5: Constant-overhead PCG for N instances of random bit-OT.

Proof. Correctness. We show correctness in two steps. First, we show that indeed the output correlation generated is bit-OT. Next, we show that the output distribution of each party individually is indistinguishable from random. Together, these imply correctness.

Towards well-formedness of the correlation, we have to show the following.

- If $b_j = 0$, then $\pi_j(y_j^0) \oplus v_j^0 = \pi_j(y_j^1) \oplus v_j^1$.
- If $b_j = 1$, then $\pi_j(y_j^0) \oplus v_j^0 = \pi_j(y_j^1) \oplus v_j^1 \oplus \pi_j(x)$.

Recall that $\vec{\mu} \in \{0,1\}^N$ is the regular noise vector defined by α, i.e.,

$$\mu_j = \begin{cases} 1 & \text{if } j = \alpha_i + \frac{N}{t} \cdot (i-1) \\ 0 & \text{else} \end{cases}.$$

Thus, for $j \in [N]$ it holds

$$\pi_j(x \cdot \mu_j) = \begin{cases} \pi_j(x) & \text{if } j = \alpha_i + \frac{N}{t} \cdot (i-1) \\ 0 & \text{else} \end{cases} = \begin{cases} \beta_i & \text{if } j = \alpha_i + \frac{N}{t} \cdot (i-1) \\ 0 & \text{else} \end{cases}.$$

By the correctness of the multi-point FSS FSS, we have

$$v_j^0 \oplus v_j^1 = \mathsf{FullEval}(0, K_0)_j \oplus \mathsf{FullEval}(1, K_1)_j = \begin{cases} \beta_i & \text{if } j = \alpha_i + \frac{N}{t} \cdot (i-1) \\ 0 & \text{else} \end{cases},$$

which yields

$$v_j^0 \oplus v_j^1 = \pi_j(x \cdot \mu_j) = \pi_j(x) \cdot \mu_j.$$

Further, recall that it holds

$$\vec{r}_0 \oplus \vec{r}_1 = \vec{r}$$

and thus

$$\vec{y}^0 \oplus \vec{y}^1 = \mathbf{G} \cdot \vec{r}_0 \oplus \mathbf{G} \cdot \vec{r}_1 = \mathbf{G} \cdot \vec{r}.$$

Now, recalling $r_j = s_j \cdot x$ for all $j \in [N]$, we have that

$$(\mathbf{G} \cdot \vec{r})_j = (\mathbf{G} \cdot \vec{s})_j \cdot x = (\vec{b} \oplus \vec{\mu})_j \cdot x,$$

where the last equality follows by the definition of $\vec{b}$. This implies

$$y_j^0 \oplus y_j^1 = (b_j \oplus \mu_j) \cdot x$$

and thus

$$\pi_j(y_j^0) \oplus \pi_j(y_j^1) = \pi_j(y_j^0 \oplus y_j^1) = \pi_j((b_j \oplus \mu_j) \cdot x) = \pi_j(x) \cdot (b_j \oplus \mu_j),$$

by the linearity of the projection map π_j. Altogether, we obtain

$$\pi_j(y_j^0) \oplus \pi_j(y_j^1) \oplus v_j^0 \oplus v_j^1 = \pi_j(x) \cdot (b_j \oplus \mu_j) \oplus \pi_j(x) \cdot \mu_j = \begin{cases} 0 & \text{if } b_j = 0 \\ \pi_j(x) & \text{if } b_j = 1 \end{cases},$$

as required.

It remains to show pseudorandomness of the individual output shares. Note that for correlations such as the bit-OT correlation, pseudorandomness of the individual output shares follows from security of the PCG. For completeness, we still give a proof of pseudorandomness in the following.

We start by showing pseudorandomness of $\mathsf{Expand}(0, \mathsf{k}_0)$. The proof proceeds via series of games.

- **Game 0**: Generate the output $\{(b_j, w_j)\}_{j \in [N]}$ according to $\mathsf{Expand}(0, \mathsf{k}_0)$.
- **Game 1**: Generate the output $\{(b_j, w_j)\}_{j \in [N]}$ by choosing $\vec{b} \xleftarrow{\$} \{0,1\}^N$ uniformly at random, otherwise proceed as in $\mathsf{Expand}(0, \mathsf{k}_0)$.
- **Game 2**: Generate the output $\{(b_j, w_j)\}_{j \in [N]}$ by choosing $\vec{b} \xleftarrow{\$} \{0,1\}^N$ and $\vec{v}_0 \leftarrow (\{0,1\}^\ell)^N$ uniformly at random, otherwise proceed as in $\mathsf{Expand}(0, \mathsf{k}_0)$.
- **Game 3**: Generate the output $\{(b_j, w_j)\}_{j \in [N]}$ by choosing $\vec{b} \xleftarrow{\$} \{0,1\}^N$ and $\vec{w} \xleftarrow{\$} \{0,1\}_{j \in [N]}$ uniformly at random.

By the $(\mathsf{Reg}_t^N(\mathbb{F}_2), \mathbf{C}, \mathbb{F}_2)$-$\mathsf{LPN}(n, N)$-assumption, we have that **Game 0** and **Game 1** are computationally indistinguishable. The indistinguishability of **Game 1** and **Game 2** is a straightforward consequence of the pseudorandomness of $\mathsf{FSS.FullEval}(0, K_0)$ (Remark 14). Finally, observe that by the pseudorandomness of G, it follows that for random input the output of $G' \colon (\{0,1\}^\kappa)^N \to \{0,1\}^N, (\vec{v}_j) \mapsto (P_j(\vec{v}_j))_{j\in[N]}$ is indistinguishable from random.[9] This implies that **Game 2** and **Game 3** are indistinguishable.

It is left to consider pseudorandomness of $\mathsf{Expand}(1, \cdot)$. As before, the proof proceeds via a sequence of games.

- **Game 0**: Generate the output $\{(w_{j,0}, w_{j,1})\}_{j\in[N]}$ according to $\mathsf{Expand}(1, \mathsf{k}_1)$.
- **Game 1**: Generate $\{(w_{j,0}, w_{j,1})\}_{j\in[N]}$ by sampling $\vec{v}_1 \xleftarrow{\$} (\{0,1\}^\ell)^N$ uniformly at random, and otherwise according to $\mathsf{Expand}(1, \mathsf{k}_1)$.
- **Game 2**: Generate $\{(w_{j,0}, w_{j,1})\}_{j\in[N]}$ by sampling $\vec{v}_1 \xleftarrow{\$} (\{0,1\}^\ell)^N$ and $\vec{w}_1 \xleftarrow{\$} \{0,1\}^N$ uniformly at random, and otherwise proceed as in $\mathsf{Expand}(1, \mathsf{k}_1)$.
- **Game 3**: Generate the output $\{(w_{j,0}, w_{j,1})\}_{j\in[N]}$ by sampling $\vec{w}_0, \vec{w}_1 \xleftarrow{\$} \{0,1\}^N$ uniformly at random.

The indistinguishability of **Game 0** and **Game 1** follows by the pseudorandomness of $\mathsf{FSS.FullEval}(1, K_1)$. The indistinguishability between **Game 1** and **Game 2** follows by the correlation-robustness of the PRG G.[10] Finally, the indistinguishability between **Game 2** and **Game 3** follows by the pseudorandomness of G, as before.

Security. It remains to show pseudorandomness of the other party's output, even given one key. More precisely, for $\sigma \in \{0,1\}$ we have to show computational indistinguishability of

$$\begin{aligned} &\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)\} \text{ and} \\ &\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_{1-\sigma} \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_{1-\sigma}), \\ &\qquad\qquad R_\sigma \xleftarrow{\$} \mathsf{RSample}(\sigma, R_{1-\sigma})\} \end{aligned}$$

where $\mathsf{RSample}$ is the reverse sampling algorithm for the bit-OT correlation.

We start by proving the claim for $\sigma = 0$. To that end, we have to show that even given $\mathsf{k}_1 = (\vec{x}, S_1, K_1)$, $\{(b_j, w_j)\}_{j\in[N]}$ is pseudorandom conditioned on $w_j = w_{j,b_j}$, where $\{(w_{j,0}, w_{j,1}\}_{j\in[N]} \leftarrow \mathsf{Expand}(1, \mathsf{k}_1)$. By correctness, it suffices to show that $\vec{b}$ is indistinguishable from random, even given $\mathsf{k}_1 = (\vec{x}, S_1, K_1)$. In the following, we first alter the key generation to achieve that k_1 contains no information about $\vec{b}$, and then use the LPN assumption to switch $\vec{b}$ to random.

- **Game 0:** Generate the keys $(\mathsf{k}_0, \mathsf{k}_1)$ by proceeding as in $\mathsf{Gen}(1^\lambda)$. Further, generate $\{(b_j, w_j)\}_{j\in[N]}$ according to $\mathsf{Expand}(0, \mathsf{k}_0)$.
- **Game 1:** Generate the keys $(\mathsf{k}_0, \mathsf{k}_1)$ by proceeding as in $\mathsf{Gen}(1^\lambda)$. Further, compute $\vec{b}$ according to $\mathsf{Expand}(0, \mathsf{k}_0)$ and set $w_j := w_{j,b_j}$ for all $j \in [N]$, where $\{(w_{j,0}, w_{j,1}\}_{j\in[N]} \leftarrow \mathsf{Expand}(1, \mathsf{k}_1)$.
- **Game 2:** Generate the keys $(\mathsf{k}_0, \mathsf{k}_1)$ by additionally sampling $\vec{\alpha}' \xleftarrow{\$} [N/t]^t$ uniformly at random and setting $\beta'_i := \pi_{\alpha_{i'} + \frac{N}{t}\cdot(i-1)}(x) \in \{0,1\}^\ell$ for each $i \in [t]$ and $\vec{\beta}' := (\beta_1, \ldots, \beta_t)$. Set $(K_0, K_1) \leftarrow \mathsf{Setup}(1^\lambda, \alpha', \beta')$, and otherwise proceed as in $\mathsf{Gen}(1^\lambda)$. To compute $\mathsf{Expand}(0, \mathsf{k}_0)$ proceed as in **Game 1**.
- **Game 3:** Generate the keys $(\mathsf{k}_0, \mathsf{k}_1)$ as in **Game 2**. Further, generate $\{(b_j, w_j)\}_{j\in[N]}$ by sampling $\vec{b} \xleftarrow{\$} \{0,1\}^N$ uniformly at random and setting $w_j := w_{j,b_j}$ for all $j \in [N]$, where $\{(w_{j,0}, w_{j,1}\}_{j\in[N]} \leftarrow \mathsf{Expand}(1, \mathsf{k}_1)$.

By the correctness of the PCG, it follows that **Game 0** and **Game 1** are identical. Further, note that the output of $\mathsf{Expand}(1, \mathsf{k}_1)$ in **Game 1** is independent from $\mathsf{FSS.FullEval}(1, K_1)$. By the security of the multi-point FSS FSS we thus have that **Game 1** and **Game 2** are computationally indistinguishable. In game **Game 2** the key k_1 contains no information about $\vec{b}$. The indistinguishability of **Game 2** and **Game 3** is thus a direct consequence of the $(\mathsf{Reg}_t^N(\mathbb{F}_2), \mathbf{C}, \mathbb{F}_2)$-$\mathsf{LPN}(n, N)$-assumption.

It is left to consider the case $\sigma = 1$. Here, we have to show that even given $\mathsf{k}_0 = (\vec{s}, S_0, \vec{\alpha}, K_0)$, $\{(w_{j,0}, w_{j,1})\}_{j\in[N]}$ is pseudorandom conditioned on $w_{j,b_j} = w_j$, where $\{(b_j, w_j)\}_{j\in[N]} \leftarrow \mathsf{Eval}(0, \mathsf{k}_0)$. Again, the proof proceeds via a series of games. Here, we crucially rely on the correlation-robustness of G.

[9] Note that this indistinguishability is even information-theoretic, since any bias in the output of a predicate could be used by a computational distinguisher to attack the underlying PRG G.

[10] Note that here we rely on correlation-robustness with shift $\Delta = (\Delta_1, \ldots, \Delta_N)$ sampled uniformly at random, so in some sense the weakest form of correlation-robustness.

– **Game 0:** Generate the keys $(\mathsf{k}_0, \mathsf{k}_1)$ by proceeding as in $\mathsf{Gen}(1^\lambda)$. Further, generate $\{(w_{j,0}, w_{j,1})\}_{j\in[N]}$ according to $\mathsf{Eval}(1, \mathsf{k}_1)$.
– **Game 1:** Generate the keys $(\mathsf{k}_0, \mathsf{k}_1)$ by proceeding as in $\mathsf{Gen}(1^\lambda)$. Compute $v_j^1 = v_j^0 \oplus \pi_j(\vec{x}) \cdot \mu_j$ for $j \in [N]$, where $\vec{v}^0$ is as computed by $\mathsf{Eval}(0, K_0)$ (instead of calling $\mathsf{FullEval}(1, K_1)$ to obtain $\vec{v}^1$). Otherwise, proceed as in $\mathsf{Eval}(1, \mathsf{k}_1)$ to generate $\{(w_{j,0}, w_{j,1})\}_{j\in[N]}$.
– **Game 2:** Generate the keys $(\mathsf{k}_0, \mathsf{k}_1)$ by additionally sampling $x' \overset{\$}{\leftarrow} \{0,1\}^\kappa$ uniformly at random, setting $\beta'_i := \pi_{\alpha_i + \frac{N}{t}\cdot(i-1)}(x') \in \{0,1\}^\ell$ for each $i \in [t]$ and $\vec{\beta}' := (\beta'_1, \dots, \beta'_t)$. Set $(K_0, K_1) \leftarrow \mathsf{Setup}(1^\lambda, \alpha', \beta')$, and otherwise proceed as in $\mathsf{Gen}(1^\lambda)$. To compute $\mathsf{Eval}(1, K_1)$ proceed as in **Game 1**.
– **Game 3:** Generate the keys $(\mathsf{k}_0, \mathsf{k}_1)$ as in **Game 2**. Further, generate $\{(w_{j,0}, w_{j,1})\}_{j\in[N]}$ by sampling $w_{j,1-b_j} \overset{\$}{\leftarrow} \{0,1\}$ uniformly at random and setting $w_{j,b_j} := w_j$ for each $j \in [N]$, where $\{(b_j, w_j)\}_{j\in[N]} \leftarrow \mathsf{Eval}(0, \mathsf{k}_0)$.

Recall from the proof of correctness that it holds $v_j^0 \oplus v_j^1 = \pi_j(x) \cdot \mu_j$, so the transition from **Game 0** to **Game 1** is perfectly indistinguishable. Since the output of $\mathsf{Expand}(1, \mathsf{k}_1)$ is independent of the output of $\mathsf{FSS.FullEval}(1, K_1)$, the transition from **Game 1** to **Game 2** is indistinguishable by the security of the multi-point FSS FSS. Finally, recall from the proof of correctness that it holds

$$\pi_j(y_j^0) \oplus \pi_j(y_j^1) \oplus v_j^0 \oplus v_j^1 = b_j \cdot \pi_j(x).$$

Setting $\Delta_j := v_j^0 \oplus \pi_j(y_j^0)$ for $j \in [N]$, we obtain that in **Game 2**, $w_{j,b}$ is computed as

$$w_{j,b} := P_j(\pi_j(y_j^1) \oplus v_j^1 \oplus b \cdot \pi_j(x)) = P_j(\Delta_j \oplus b_j \cdot \pi_j(x) \oplus b \cdot \pi_j(x)).$$

We thus obtain

$$w_{j,b} = \begin{cases} P_j(\Delta_j) & \text{if } b = b_j \\ P_j(\Delta_j \oplus \pi_j(x)) & \text{if } b \neq b_j \end{cases}.$$

The transition from **Game 2** and **Game 3** is thus indistinguishable as a consequence of the correlation-robustness of G, recalling that in both games the key k_0 is independent of the seed x. □

Lemma 31. *The* $\mathsf{PCG.Gen}$ *algorithm in Fig. 5 has circuit size* $O(\kappa \cdot n + C_{\mathsf{FSS.Setup}})$. *Furthermore, if* $C^\sigma_{\mathsf{FSS}}(\lambda)$ *is the cost of* $\mathsf{FSS.FullEval}(\sigma, \cdot)$ *and* C_P *is an upper bound on the cost of evaluating one predicate in the local PRG* G*, then the* $\mathsf{PCG.Expand}(\sigma, \cdot)$ *phase has circuit size*

$$\ell dN + C^\sigma_{\mathsf{FSS}}(\lambda) + (1-\sigma)dN + (1+\sigma)(C_P + 1)N.$$

So, if FSS *has constant overhead then* $\mathsf{PCG.Expand}$ *has constant overhead.*

Proof. For key generation, generating the secret shares $\vec{r}_0, \vec{r}_1$ requires $O(n \cdot \kappa)$ operations. The remainder of the setup is dominated by $\mathsf{FSS.Setup}$, giving $O(\kappa \cdot n + C_{\mathsf{FSS.Setup}}(\lambda))$.

For expansion, the cost derivation is as follows.

– Computing each entry of $\vec{b}$ (for $\sigma = 0$) can be done with d XORs, for a total of dN gates.
– Since one only has to compute the ℓ-bit projections π_j of $\vec{y}^0$, $\vec{y}^1$ (as explained in the protocol description), these cost at most ℓdN XOR gates.
– Computing $\vec{v}^\sigma$ costs $C^\sigma_{\mathsf{FSS}}(\lambda) \cdot N$ gates.
– Each w_j (for $\sigma = 0$) or $w_{j,0}, w_{j,1}$ ($\sigma = 1$), can be computed with $C_P + 1$ gates, resulting in either $(C_P + 1)N$ (for $\sigma = 0$) or $2(C_P + 1)N$ ($\sigma = 1$) gates for the last step.

□

We can instantiate the FSS construction with the naïve "square-root" construction of known-index DPF from Section 3.1. This gives $C^\sigma_{\mathsf{FSS}}(\lambda) \leq C_{\mathsf{prg}}(\ell) + (1-\sigma)2\ell$. With regular noise, the setup cost of the FSS is $O(t\lambda\sqrt{N/t}) = O(\lambda\sqrt{Nt})$. For the PCG to be sublinear, we therefore get the constraint that $\kappa n + \lambda\sqrt{Nt} = o(N)$.

Based on the above analysis, we now obtain our main result on maliciously secure bit-OT, by replacing $\mathsf{PCG.Gen}$ with a secure 2-PC protocol.

Corollary 32. *Suppose OT exists. Suppose the* $(\mathsf{Reg}^N_t(\mathbb{F}_2), \mathbf{C}, \mathbb{F}_2)$*-*$\mathsf{LPN}(n, N)$*-assumption holds for some* n, t, N *and matrix* $\mathbf{G}$ *with constant sparsity* d*, and suppose there exists a correlation-robust* ℓ*-local PRG for constant* ℓ *that stretches* $N^{1-\varepsilon}$ *to* N *bits for some* $\varepsilon \in (0,1)$*, where* $19\varepsilon/20 \geq \frac{20}{d+18}$ *and* $0.9\cdot(19\varepsilon/20) + 9\varepsilon/10 > 1$*. Then, there exists a protocol for securely computing* N *instances of random bit-OT with malicious security,* $o(N)$ *communication and an average, amortized per-party computation of* $\ell(d+1) + \frac{d}{2} + C_{\mathsf{prg}}(\ell) + \frac{3}{2}(C_P + 1)$ *Boolean gates per OT.*

Proof. Assuming OT and using standard 2-PC protocols like [IPS08], there is a polynomial p such that for all λ and circuits C, there is a malicious 2-PC protocol that securely computes C with computation complexity $O(|C|)\cdot p(\lambda)$. Based on Lemma 31 and plugging in the square-root FSS construction, we obtain a protocol that securely computes the $\mathsf{PCG.Gen}$ algorithm from Fig. 5 with complexity $(\kappa n + \lambda\sqrt{Nt})\cdot p(\lambda)$. Following [BCG+19b, Theorem 19], by running the 2-PC protocol and then locally evaluating $\mathsf{PCG.Expand}$, the resulting protocol securely realizes the functionality for N instances of random bit-OT.

We show how to choose parameters such that the complexity of the 2-PC phase is sublinear in N for sufficiently large N. This means the $p(\lambda)$ overhead of 2-PC amortizes and the total computational cost is dominated by the expand phase. With $\kappa = N^{1-\varepsilon}$, we set $n = N^{19\varepsilon/20}$ and $t = N^{9\varepsilon/10}$. Further, we obtain complexities of $\kappa n = N^{1-\varepsilon/20}$ and $\lambda\sqrt{Nt} = \lambda N^{\frac{1}{2}+\frac{9}{20}\varepsilon}$. These are both sublinear in N. Regarding security of primal-LPN, note that by assumption we have $n \geq N^{20/(d+18)}$, which is enough to give dual distance $D = \Theta(n^{9/10})$. We also have $tn^{9/10} = \Omega(N\lambda)$ for large enough N by assumption, giving exponential security in λ.

Finally, to compute the computational complexity of $\mathsf{PCG.Expand}$, we plug in the cost of the square-root FSS construction to the formula from Lemma 31, and average the result over the two parties $\sigma = 0$ and $\sigma = 1$. This is possible as random bit-OT is symmetric, so the parties can run two instances of the protocol of size $N/2$, reversing the roles of sender and receiver. □

Concrete Complexity. We now estimate the constant overhead of our construction, at least asymptotically as N grows large. We need to choose the LPN degree d and ℓ-local correlation-robust PRG, which determines the predicate cost C_P and the PRG seed size $\kappa = N^{1-\varepsilon}$ bits. We also need to instantiate the PRG used in the FSS scheme, for which we use a 5-local PRG, giving $C_{\mathsf{prg}}(\ell) = C_{P_5}\ell = 4\ell$ (unlike the other PRG, this one only needs to have constant stretch, since it can be used iteratively). Using the $\mathsf{XOR\text{-}MAJ}_{4,5}$ predicate (see Section C.1) in our PCG, we have $\ell = 9$, $C_P = 17$ and a plausible stretch of $\kappa^{2.49}$, giving $\varepsilon = 1 - 1/2.49$. This satisfies $19\varepsilon/20 > \frac{20}{d+18}$ for $d = 18$, and furthermore that $0.9\cdot(19\varepsilon/20) + 9\varepsilon/10 > 1$. Note further that $19\varepsilon/20 > 0.5$, so $n = N^{19\varepsilon/20} = \omega(\sqrt{N})$, ruling out the Arora-Ge attack [AG11]. Furthermore $t = N^{9\varepsilon/10} = o(n)$, which is necessary for building on LPN with regular noise. Overall, we get a cost of 243 gates per party.

Remark 33 (Iterative constant overhead). If the FSS scheme supports single evaluation Eval with constant cost $c \in O(1)$, the above construction yields a PCG with iterative constant overhead, i.e., where a single bit-OT can be computed at constant cost. This property in fact already holds of the square-root FSS construction when instantiated with a local PRG.

Remark 34 (Building on LPN with standard noise). To build on LPN with the more standard Bernoulli noise, one has to replace the t-regular FSS by a more general multi-point FSS. Known-index multi-point FSS with constant overhead can be obtained generically from single-point FSS with constant overhead via batch codes. For details we refer to [BCGI18].

5 Constant-Overhead PCG for OT from Dual LPN

In this section we provide a PCG for OT with constant computational overhead based on a dual-LPN assumption. This will allow us to achieve an arbitrary polynomial stretch.

Repeat-Accumulate (RA) Codes. We consider the following dual code generation procedure, which essentially outputs generator matrices for *repeat-accumulate (RA) codes.*

Definition 35. *Let $d, N \in \mathbb{N}$ with $d \geq 3$ and let $M = dN$. A* repeat-accumulate (RA) *matrix $\mathbf{H}$ is a matrix of the form $\mathbf{H} = \mathbf{BA}$, where $\mathbf{B} \in \{0,1\}^{N\times M}$ has exactly d nonzero entries per row and 1 nonzero entry per column and $\mathbf{A} \in \{0,1\}^{M\times M}$ is an accumulator matrix which has 1's on and below the main diagonal.*

The code $\{\vec{u}^\top \cdot \mathbf{BA} : \vec{u} \in \{0,1\}^N\}$ is well-studied in coding theory (it is called a repeat-accumulate (RA) code). In particular, for fixed $\gamma > 0$ it is known that they achieve minimum distance $D = M^{1-2/d-\gamma}$ with good probability over a random choice of $\mathbf{B}$. This means that if $\vec{e} \overset{\$}{\leftarrow} \mathsf{Reg}_t^M(\{0,1\})$, recalling that the bias of the dot-product between a vector of weight D and $\vec{e}$ is at most $e^{-2tD/M}$ we will require $t \geq \ln 2 \cdot \lambda \cdot M^{2/d+\gamma}$.

Unfortunately, the failure probability is not negligible (an inspection of the proof of [KZCJ07, Theorem 1] shows that it is roughly of the order $M^{-\gamma d/2}$). Thus, as before we will assume access to a *explicitly generated* RA matrix $\mathbf{H} = \mathbf{BA}$ and assume that the dual $(\mathsf{Reg}_t^M(\{0,1\}), \mathbf{H}, \{0,1\}\text{-}\mathsf{LPN}(N, dN)$ holds with respect to it.[11]

Construction $\mathsf{PCG}_{\mathsf{OT}}^{\mathsf{dual}}$

PARAMETERS:

- Security parameter $\lambda \in \mathbb{N}$, matrix parameters $M = M(\lambda), N = N(\lambda) \in \mathbb{N}$ with $M = dN$ for constant matrix sparsity parameter $d \in \mathbb{N}$, noise weight $t = t(\lambda) \in \mathbb{N}$, local PRG input length $\kappa = \kappa(\lambda) \in \mathbb{N}$, constant locality $\ell \in \mathbb{N}$.
- An RA matrix $\mathbf{H} = \mathbf{BA} \in \{0,1\}^{N \times M}$ for which the dual $(\mathsf{Reg}_t^M(\{0,1\}), \mathbf{H}, \{0,1\})\text{-}\mathsf{LPN}(N, dN)$ holds. Let S_j for $j \in [N]$ denote the support of the j-th row of $\mathbf{B}$ (each of which has size d) and for $i \in [M]$ let $\tau(i) \in [N]$ denote the nonzero coordinate of the i-th column of $\mathbf{B}$. //Note that $S_j = \{i \in [M] : \tau(i) = j\} = \tau^{-1}(j)$.
- A constant overhead regular known-index projected-payload t-interval FSS $\mathsf{FSS} = (\mathsf{Setup}, \mathsf{Eval})$ over domain $[M]$ with output bit length κ and projected output length ℓ.
- A correlation-robust ℓ-local PRG $G\colon \{0,1\}^\kappa \to \{0,1\}^N$ with $G(x) = P_1(\pi_1(x)) \| \ldots \| P_N(\pi_N(x))$ for all $x \in \{0,1\}^\kappa$.

CORRELATION: Outputs N tuples $((b, w), (w_0, w_1))$, where b, w_0, w_1 are random bits and $w = w_b$.

GEN:

- Pick a *local PRG seed* $x \xleftarrow{\$} \{0,1\}^\kappa$ at random.
- Choose *regular noise positions* $\vec{\alpha} \xleftarrow{\$} [M/t]^t$ at random.
- Set $\beta := x \in \{0,1\}^\kappa$.
- Denote $\vec{\psi} = (\pi_{\tau(1)}, \ldots, \pi_{\tau(M)})$.
- Set $(K_0, K_1) \leftarrow \mathsf{FSS.Setup}(1^\lambda, (\vec{\alpha}, \beta, \vec{\psi}))$.
- Set $\mathsf{k}_0 := (\vec{\alpha}, K_0)$ and $\mathsf{k}_1 := (x, K_1)$ and output $(\mathsf{k}_0, \mathsf{k}_1)$.

EXPAND: On input $(\sigma, \mathsf{k}_\sigma)$:

1. If $\sigma = 0$, parse k_0 as $(\vec{\alpha}, K_0)$ and proceed as follows:
   - Let $\vec{a} \in \{0,1\}^M$ be the regular interval noise vector defined by $\vec{\alpha}$, i.e., a_i is equal to the parity of $|\{\iota \in [t] : \alpha_\iota + (M/t) \cdot (\iota - 1) \le i\}|$.
   - Compute $\vec{b} := \mathbf{B} \cdot \vec{a}$.
   - Compute $\vec{v}^0 \leftarrow \mathsf{FSS.FullEval}(0, K_0) \in (\{0,1\}^\ell)^M$.
   - For $j \in [N]$ compute $w_j := P_j\left(\bigoplus_{i \in S_j} v_i^0\right)$.
   - Output $\{(b_j, w_j)\}_{j \in [N]}$.
2. If $\sigma = 1$, parse k_1 as (x, K_1) and proceed as follows:
   - Compute $\vec{v}^1 \leftarrow \mathsf{FSS.FullEval}(1, K_1) \in (\{0,1\}^\ell)^M$.
   - For $j \in [N]$, $b \in \{0,1\}$ compute $w_{j,b} := P_j\left(\left(\bigoplus_{i \in S_j} v_i^1\right) \oplus b \cdot \pi_j(x)\right)$.
   - Output $\{(w_{j,0}, w_{j,1})\}_{j \in [N]}$.

Fig. 6: Constant-overhead PCG for N instances of random bit-OT.

Theorem 36. *Let $N = N(\lambda), t = t(\lambda), \kappa = \kappa(\lambda) \in \mathbb{N}$, let $\ell, d \in \mathbb{N}$ be constants, let $M = dN$ and let $\mathbf{H} = \mathbf{BA}$ be an $N \times M$ repeat-accumulate matrix. If the dual $(\mathsf{Reg}_t^M(\{0,1\}), \mathbf{H}, \{0,1\})\text{-}\mathsf{LPN}(N, M)$-assumption holds, if $G\colon \{0,1\}^\kappa \to \{0,1\}^N$ is a correlation-robust ℓ-local PRG, and if FSS is a known-index projected-payload t-interval FSS, then the PCG as defined in Figure 6 is a constant-overhead PCG for generating N instances of the bit-OT correlation.*

Proof. Correctness. As in the primal case, proving correctness consists of two steps. First, a demonstration that the output correlation is indeed bit-OT; second, that each parties bits individually look uniformly random.

[11] Instead of RA codes we could have used a code of Tillich and Zémor [TZ06]; however the effect on the computational complexity is essentially nil, so we have chosen to just present the RA-code based construction.

For the first step, for each $j \in [N]$ we need to show

$$\bigoplus_{i \in S_j} v_i^0 = \bigoplus_{i \in S_j} v_i^1 \oplus b_j \cdot \pi_j(x),$$

which is the same as

$$\bigoplus_{i \in S_j} v_i^0 \oplus \bigoplus_{i \in S_j} v_i^1 = b_j \cdot \pi_j(x).$$

Recall $\vec{a} \in \{0,1\}^M$ denotes the interval noise vector defined by $\vec{\alpha}$, i.e., a_i is the parity of $|\{\iota \in [t] : \alpha_\iota + (M/t) \cdot (\iota - 1) \leq i\}|$. Thus, by correctness of the FSS scheme, we have

$$\begin{aligned}\bigoplus_{i \in S_j} v_i^0 \oplus \bigoplus_{i \in S_j} v_i^1 &= \bigoplus_{i \in S_j} v_i^0 \oplus v_i^1 = \bigoplus_{i \in S_j} \mathsf{FullEval}(0, K_0)_i \oplus \mathsf{FullEval}(1, K_1)_i \\ &= \bigoplus_{i \in S_j} a_i \cdot \pi_{\tau(i)}(x) = \bigoplus_{i \in S_j} a_i \cdot \pi_j(x) = \pi_j(x) \cdot \bigoplus_{i \in S_j} a_i = \pi_j(x) \cdot b_j.\end{aligned}$$

Thus, the output correlation is indeed a bit-OT.

Now, we show that each of the output shares are individually pseudorandom. For the pseudorandomenss of $\mathsf{Expand}(0, \mathsf{k}_0)$, consider the following sequence of games.

- **Game 0:** Generate the output $\{(b_j, w_j)\}_{j \in [N]}$ as in $\mathsf{Expand}(0, \mathsf{k}_0)$.
- **Game 1:** Generate the output $\{(b_j, w_j)\}_{j \in [N]}$ by choosing $\vec{b} \overset{\$}{\leftarrow} \{0,1\}^N$ uniformly at random, otherwise proceed as in $\mathsf{Expand}(0, \mathsf{k}_0)$.
- **Game 2:** Generate the output $\{(b_j, w_j)\}_{j \in [N]}$ by choosing $\vec{b} \overset{\$}{\leftarrow} \{0,1\}^N$ and $\vec{v}^0 \overset{\$}{\leftarrow} (\{0,1\}^\ell)^M$ uniformly at random, but otherwise proceed as in $\mathsf{Expand}(0, \mathsf{k}_0)$.
- **Game 3:** Generate the output $\{(b_j, w_j)\}_{j \in [N]}$ by choosing $\vec{b} \overset{\$}{\leftarrow} \{0,1\}^N$ and $\vec{w} \overset{\$}{\leftarrow} \{0,1\}^N$.

The computational indistinguishability of **Game 0** and **Game 1** follows from the dual $(\mathsf{Reg}_t^M(\{0,1\}), \mathbf{H}, \{0,1\})$-$\mathsf{LPN}(N, M)$ assumption. The indistinguishability of **Game 1** and **Game 2** is a direct consequence of the pseudorandomness of $\mathsf{FSS.FullEval}(0, K_0)$ (Remark 14). For the indistinguishability of **Game 2** and **Game 3**, first note that if $\vec{v}^0 \overset{\$}{\leftarrow} (\{0,1\}^\ell)^M$ uniformly at random, then since $S_1, \ldots, S_N$ give a partition of $[M]$ each $\bigoplus_{i \in S_j} v_i^0$ is uniformly random. By the pseudorandomness of G, it must be that the string $\left(P_j\left(\bigoplus_{i \in S_j} v_i^0\right)\right)_{j \in [N]}$ statistically indistinguishable from random (as a noticeable bias would give a distinguisher for G). It follows that **Game 2** and **Game 3** are indistinguishable.

We now consider $\mathsf{Expand}(1, \mathsf{k}_1)$, and again demonstrate its pseudorandomness via a sequence of games.

- **Game 0:** Generate the output $\{(w_{j,0}, w_{j,1})\}_{j \in [N]}$ according to $\mathsf{Expand}(1, \mathsf{k}_1)$.
- **Game 1:** Generate $\{(w_{j,0}, w_{j,1})\}_{j \in [N]}$ by sampling $v^1 \overset{\$}{\leftarrow} (\{0,1\}^\ell)^M$ uniformly at random, and otherwise proceeding as in $\mathsf{Expand}(1, \mathsf{k}_1)$.
- **Game 2:** Generate the output $\{(w_{j,0}, w_{j,1})\}_{j \in [N]}$ by sampling $v^1 \overset{\$}{\leftarrow} (\{0,1\}^\ell)^M$ and $\vec{w}_1 \overset{\$}{\leftarrow} \{0,1\}^N$ uniformly at random, and otherwise proceeding as in $\mathsf{Expand}(1, \mathsf{k}_1)$.
- **Game 3:** Generate the output $\{(w_{j,0}, w_{j,1})\}_{j \in [N]}$ by sampling $\vec{w}_0, \vec{w}_1 \overset{\$}{\leftarrow} \{0,1\}^N$ uniformly at random.

The computational indistinguishability of **Game 0** and **Game 1** follows from the pseudorandomness of $\mathsf{FSS.FullEval}(1, K_1)$ (Remark 14). The computational indistinguishability of **Game 1** and **Game 2** follows from the correlation-robustness of the PRG G, with the shift $(\Delta_1, \ldots, \Delta_N)$ chosen as $\Delta_j = \pi_j(x)$. Finally, the (even statistical) indistinguishability of **Game 2** and **Game 3** follows from the fact that the predicates P_j cannot have noticeable bias, since for each j, $\bigoplus_{i \in S_j} v_i^1$ is uniformly random.

Security. We now verify the pseudorandomness of the other party's output, even given one key. That is, for each $\sigma \in \{0,1\}$, we show the computational indistinguishability of the following distributions:

$$\begin{aligned}&\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \overset{\$}{\leftarrow} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)\} \text{ and} \\ &\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \overset{\$}{\leftarrow} \mathsf{PCG.Gen}(1^\lambda), R_{1-\sigma} \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_{1-\sigma}), \\ &\qquad\qquad R_\sigma \overset{\$}{\leftarrow} \mathsf{RSample}(\sigma, R_{1-\sigma})\}.\end{aligned}$$

Above, $\mathsf{RSample}$ is the reverse sampling algorithm for the bit-OT correlation.

Suppose first $\sigma = 0$. To prove the above claim, we must show that given $\mathsf{k}_1 = (x, K_1)$, the output $\{(b_j, w_j)\}_{j\in[N]} \leftarrow \mathsf{Expand}(0, \mathsf{k}_0)$ is pseudorandom conditioned on $w_j = w_{j,b_j}$, where $\{(w_{j,0}, w_{j,1})\}_{j\in[N]} \leftarrow \mathsf{Expand}(1, \mathsf{k}_1)$. Given the conditioning, this amounts to showing that $\vec{b}$ is indistinguishable from random even given (x, K_1). To do this, we again use a sequence of games.

– **Game 0:** Generate the keys $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{Gen}(1^\lambda)$ and generate $\{(b_j, w_j)\}_{j\in[N]}$ according to $\mathsf{Expand}(0, \mathsf{k}_0)$.
– **Game 1:** Generate the keys $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{Gen}(1^\lambda)$ and then generate $\vec{b}$ according to $\mathsf{Expand}(0, \mathsf{k}_0)$ and generate $\vec{w}$ by setting $w_j = w_{j,b_j}$ for all $j \in [N]$ where we have generated $\{(w_{j,0}, w_{j,1})\}_{j\in[N]} \leftarrow \mathsf{Expand}(1, \mathsf{k}_1)$.
– **Game 2:** To generate the keys $(\mathsf{k}_0, \mathsf{k}_1)$, sample an independent $\vec{\alpha}' \xleftarrow{\$} [M/t]^t$, and let $(K_0, K_1) \leftarrow \mathsf{FSS.Setup}(1^\lambda, (\vec{\alpha}', \beta, \vec{\psi}))$. Subsequently output $\mathsf{k}_0 = (\vec{\alpha}, K_0)$ and $\mathsf{k}_1 = (x, K_1)$. Then compute $\vec{b}$ and $\vec{w}$ as in **Game 1**.
– **Game 3:** Generate keys $(\mathsf{k}_0, \mathsf{k}_1)$ as in **Game 2**. Then, sample $\vec{b} \xleftarrow{\$} \{0,1\}^N$ uniformly at random. Finally, compute $\vec{w}$ as in **Game 1**.

First, the correctness of the PCG promises that **Game 0** and **Game 1** are identical. Next, note that the secrecy assumption of the FSS scheme tells us that even given β the distributions of the keys (K_0, K_1) in **Game 1** and **Game 2** are indistinguishable, as the corresponding Leak_1's are the same. This implies that **Game 1** is computationally indistinguishable from **Game 2**. Lastly, the computational indistinguishability of **Game 2** and **Game 3** follows from the dual $(\mathsf{Reg}_t^M(\{0,1\}), \mathbf{H}, \{0,1\})\text{-}\mathsf{LPN}(N, M))$-assumption, using the fact that k_1 is independent of $\vec{\alpha}$ in both of these games.

Next, suppose $\sigma = 1$. In this case, we have to show that even given $\mathsf{k}_0 = (\vec{\alpha}, K_0)$ the output $\{(w_{j,0}, w_{j,1})\}_{j\in[N]} \leftarrow \mathsf{Expand}(1, \mathsf{k}_1)$ is pseudorandom conditioned on $w_j = w_{j,b_j}$, where $\{(b_j, w_j)\}_{j\in[N]} \leftarrow \mathsf{Expand}(0, \mathsf{k}_0)$. Given the conditioning, this amounts to showing that $w_{j,1-b_j}$ is indistinguishable from random even given $(\vec{\alpha}, K_0)$. In this case we consider the following games.

– **Game 0:** Generate the keys $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{Gen}(1^\lambda)$ and generate $\{(w_{j,0}, w_{j,1})\}_{j\in[N]}$ according to $\mathsf{Expand}(1, \mathsf{k}_1)$.
– **Game 1:** Generate the keys $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{Gen}(1^\lambda)$. To compute $\vec{v}^1$ we first compute $\vec{v}^0 \leftarrow \mathsf{FSS.FullEval}(0, K_0)$ and then set $v_i^1 = v_i^0 \oplus (a_i \cdot \pi_{\sigma(i)}(x))$ for all $i \in [M]$. Otherwise, proceed as in $\mathsf{Expand}(1, \mathsf{k}_1)$ to generate $\{(w_{j,0}, w_{j,1})\}_{j\in[N]}$.
– **Game 2:** To generate the keys $(\mathsf{k}_0, \mathsf{k}_1)$, sample an independent $x' \xleftarrow{\$} \{0,1\}^\kappa$, define $\beta' := x'$, and let $(K_0, K_1) \leftarrow \mathsf{FSS.Setup}(1^\lambda, (\vec{\alpha}, \beta', \vec{\psi}))$. Subsequently output $\mathsf{k}_0 = (\alpha, K_0)$ and $\mathsf{k}_1 = (x, K_1)$. Then compute $\{(w_{j,0}, w_{j,1})\}_{j\in[N]}$ as in **Game 1**.
– **Game 3:** Generate keys $(\mathsf{k}_0, \mathsf{k}_1)$ as in **Game 2**. Then, after generating $\{(b_j, w_j)\}_{j\in[N]} \leftarrow \mathsf{Expand}(0, \mathsf{k}_0)$, generate $\{(w_{j,0}, w_{j,1})\}_{j\in[N]}$ by setting $w_{j,b_j} := w_j$ and sampling $w_{j,1-b_j} \xleftarrow{\$} \{0,1\}$ uniformly at random for all $j \in [N]$.

First, we note that **Game 0** and **Game 1** are indistinguishable as we have $v_i^0 \oplus v_i^1 = a_i \cdot \pi_{\sigma(i)}(x)$ for all $i \in [M]$. Next, note that by the secrecy assumption of the FSS scheme we have that keys generated via $\mathsf{Setup}(1^\lambda, f_{<\vec{\alpha}}^{\beta,\vec{\pi}^\tau})$ are indistinguishable from those generated via $\mathsf{FSS.Setup}(1^\lambda, f_{<\vec{\alpha}}^{\beta',\vec{\pi}^\tau})$, as the corresponding Leak_0's are the same. Thus, we have that **Game 1** and **Game 2** are indistinguishable. Finally, for the computational indistinguishability of **Game 2** and **Game 3**, we use the correlation-robustness of G. For $j \in [N]$ set $\Delta_j := \bigoplus_{i\in S_j} v_i^0$. By the definition of $\vec{v}^0$ and $\vec{v}^1$ (see **Game 1**) we have $\Delta_j = \left(\bigoplus_{i\in S_j} v_i^1\right) \oplus b_j \cdot \pi_j(x)$ (*cf.* the earlier proof of correctness). Under this definition we find

$$w_{j,b} = P_j\left(\left(\bigoplus_{i\in S_j} v_i^1\right) \oplus b \cdot \pi_j(x)\right) = \begin{cases} P_j(\Delta_j) & \text{if } b_j = b \\ P_j(\Delta_j \oplus \pi_j(x)) & \text{if } b_j \neq b \end{cases}.$$

Using that k_0 is independent of x, the correlation robustness of G yields the computational indistinguishability of **Game 2** and **Game 3**, as desired. □

Lemma 37. *The* $\mathsf{PCG.Gen}$ *algorithm in Fig. 6 has circuit size* $O(t \log(N/t) + C_{\mathsf{FSS.Setup}}(\lambda))$. *Furthermore, if* $C_{\mathsf{FSS}}^\sigma(\lambda)$ *is the cost of* $\mathsf{FSS.FullEval}(\sigma, \cdot)$ *and* C_P *is an upper bound on the cost of evaluating one predicate in the local PRG* G*, then the* $\mathsf{PCG.Expand}(\sigma, \cdot)$ *phase has circuit size*

$$C_{\mathsf{FSS}}^\sigma(\lambda) + (1-\sigma)(2Nd - 1) + (d\ell + C_P)N + \sigma((d+1)\ell + C_P)N\ .$$

Proof. For the key generation step, we have the following costs:

– Sampling $\vec{\alpha} \xleftarrow{\$} [M/t]^t$ takes $O(t \log(M/t)) = O(t \log(N/t))$ operations.
– The setup step for the FSS keys takes time $C_{\mathsf{FSS.Setup}}(\lambda)$ by assumption.

For the expansion steps, we have the following costs. When $\sigma = 0$:

– Determining $\vec{a}$ can be done with $M - 1 = Nd - 1$ XOR operations.
– Computing $\vec{b} := \mathbf{B} \cdot \vec{a}$ takes Nd XOR operations.
– $\mathsf{FSS.FullEval}(0, K_0)$ has a circuit of size $C^{\sigma}_{\mathsf{FSS}}(\lambda)$ by assumption.
– Lastly, each w_j can be computed with d ℓ-bit XORs followed by a computing the predicate P_j, so computing $\vec{w}$ can be done in time $(d\ell + C_P)N$ bit operations.

When $\sigma = 1$, we may remove the costs of the first two steps. The next cost is $C^{\sigma}_{\mathsf{FSS}}(\lambda)$ for the $\mathsf{FSS.FullEval}(1, K_1)$ step. Lastly computing all the $w_{j,0}$ takes $(d\ell + C_P)N$ bit operations as before, whereas computing all the $w_{j,1}$ takes $((d+1)\ell + C_P)N$ operations. □

Corollary 38. *Suppose OT exists. Suppose the* $(\mathsf{Reg}^M_t(\mathbb{F}_2), \mathbf{H}, \mathbb{F}_2)$*-*$\mathsf{LPN}(N, M)$*-assumption holds for some RA matrix* $\mathbf{H}$ *with* $M = dN$ *for constant integer* d*, and suppose there exists a correlation-robust* ℓ*-local PRG for constant* ℓ *that stretches* $N^{1-\varepsilon}$ *to* N *bits for some* $\varepsilon > 0$*. Then, there exists a protocol for securely computing* N *instances of random bit-OT with malicious security,* $o(N)$ *communication and an average, amortized per-party computation of*

$$\frac{3}{2}d\ell + d + \frac{5\ell}{2} + \frac{3}{2}C_P + C_{\mathsf{prg}}(2\ell) + \frac{1}{\kappa}C_{\mathsf{prg}}(2\lambda + \kappa) + 3$$

Boolean gates per OT. In particular, if $C_{\mathsf{prg}}(k) = O(k)$ *for integer* k*, then the amortized per-party computation is constant.*

This proof is similar to that of Corollary 32.

Proof. Assuming OT and using standard 2-PC protocols like [IPS08], there is a polynomial p such that for all λ and circuits C, there is a malicious 2-PC protocol that securely computes C with computation complexity $O(|C|) \cdot p(\lambda)$. Using Lemma 37, $|C|$ is of order $O(t \log(N/t) + C_{\mathsf{FSS.Setup}}(\lambda))$. Our construction of a multi-interval projected-payload FSS from Section 3.3 gives $C_{\mathsf{FSS.Setup}}(\lambda)) = O(\log(N/\kappa)(\lambda + \kappa))$. Thus we obtain a protocol securely computing $\mathsf{PCG.Gen}$ from Figure 6 with complexity $O(\log(N/\kappa)(\lambda + \kappa) + t \log(N/t))p(\lambda)$, after which the parties can locally run $\mathsf{PCG.Expand}$: this securely realizes the N-instance of random bit-OT functionality.

Recall that, for $\gamma > 0$, we may take $t = M^{d/2+\gamma}$ to get $Dt = \omega(N)$, where $D = M^{1-d/2-\gamma/2}$ is achievable as distance for the matrix generated by $\mathbf{H}$. This yields exponential in N security against linear attacks. We also take $\kappa = N^{1-\epsilon}$; these settings imply $O(\log(N/\kappa)(\lambda + \kappa) + t \log(N/t)) = o(N) \cdot \lambda$, i.e., the total computational cost is sublinear in N and therefore dominated by the expansion phase for large enough N.

To obtain the computational complexity of $\mathsf{PCG.Expand}$, we again plug in the computation cost, $C_{\mathsf{FSS}}(\lambda)^{\sigma}$, of our multi-interval projected-payload FSS from Section 3.3, into Lemma 37. $C_{\mathsf{FSS}}(\lambda)^{\sigma}$ in turn depends on the costs of the known-index distributed point function (DPF) from Subsection 3.1 (the logarithmic construction with truncation) and the known-index interval FSS (or distributed comparison function, or DCF for short) from Subsection 3.2. The complexity of the DPF is at most $C_{\mathsf{prg}}(2\ell) + (1-\sigma)4\ell$. For the DCF, note that it is over a domain of size N/κ, so the amortized cost is

$$\frac{1}{\kappa}(C_{\mathsf{prg}}(2\lambda + \ell) + 2\kappa(1-\sigma)(3\kappa + 4\lambda)) = \frac{C_{\mathsf{prg}}(2\lambda + \kappa)}{\kappa} + 2 + (1-\sigma)\left(3 + 4\frac{\lambda}{\kappa}\right) .$$

For large enough N (and hence, κ) the $4\frac{\lambda}{\kappa}$ term is negligible. Combining these values with the other costs from Lemma 37 yields the per-party cost, which we can again average using the same trick as before (each party is the receiver for $N/2$ instances and the sender in the remaining). □

Concrete Complexity. Choose $d = 3$. Now we choose the P_5 predicate for the local PRG so that $\ell = 5$ and $C_P = 4$. Further we use a constant stretch PRG for the FSS satisfying $C_{\mathsf{prg}}(\ell) = 4\ell$. Asymptotically we then compute 91 bit operations per output, beating the primal construction. Furthermore, note with this value of d we are stretching roughly $N^{2/3}$ bits to N bits (we can choose $t = M^{2/3+\varepsilon}$). For general d we can get stretch $(dN)^{2/d+\varepsilon}$ bits to N bits, yielding arbitrary polynomial stretch (although the complexity per OT output does increase commensurately).

6 Beyond Oblivious Transfer

In this section we discuss applications of our main result to constant-overhead implementations of other secure computation tasks. For simplicity, we refer here only to the two-party case, though most of the results in this section apply also to MPC with a constant number of parties with the same asymptotic cost. We will also refer to security against malicious parties by default.

One of the main open questions about the asymptotic complexity of cryptography is the possibility of securely computing Boolean circuits with constant computational overhead. While in the semi-honest model such a result can be based on local PRGs [IKOS08], extending this to the malicious model was posed as an open question.[12] The overhead of the best known protocols grows polylogarithmically with the security parameter and the circuit size [DIK10].

Our main result allows us to make progress on this question, by obtaining partial positive results and reducing the general question to simpler questions.

6.1 General Protocols with Relaxed Security

Our work gives the first constant-overhead protocol for (malicious bit-)OT. However, extending this to general functionalities is challenging. While it is well-known that OT is complete for secure computation [Kil88, IPS08], the best known protocols for Boolean circuits in the OT-hybrid model have polylogarithmic overhead [DIK10, GIW16]. In contrast, in the *semi-honest* OT-hybrid model, a simple "textbook" protocol [GMW87,Gol09], commonly referred to as the (semi-honest) *GMW protocol*, achieves perfect security with a small constant overhead.

A key observation from [GIP$^+$14] is that this textbook protocol actually achieves a nontrivial notion of security even against malicious parties: it is secure up to *additive attacks*. For Boolean circuits, this means that the adversary's attack capability is limited to choosing a subset of the circuit wires that are toggled. This is formalized by modifying the ideal functionality to take from the adversary an additional input bit for each wire, specifying whether to insert a NOT gate into the middle of the wire. Combining this with a standard composition theorem [Can00, Gol09], we get the following application of our main result.

Theorem 39 (Constant overhead with additive attacks). *Suppose there exists a constant-overhead OT protocol (with security against malicious parties). Then there exists a constant-overhead protocol for evaluating Boolean circuits with security up to additive attacks.*

Additive attacks can render security meaningless for some applications. For instance, capturing a zero-knowledge proof as a secure computation of the verification predicate, a malicious prover can make the verifier accept a false statement by simply toggling the final decision bit.

In some other cases, however, security up to additive attacks is still meaningful. Consider a secure computation task that has long inputs and a short output, such as applying a complex search query to a big database or a data-mining algorithm to the union of two databases. In such cases, it is often hard to reason about the security features of an ideal-model implementation, except for the syntactic guarantee that a malicious party can only learn a small amount of information about the honest party's input. The same kind of guarantee is given by a protocol with security up to additive attacks. In contrast, applying the constant-overhead semi-honest protocol from [IKOS08] to such a functionality may allow a malicious party to learn the entire input of the honest party.

6.2 Leveraging Perfect Security

Consider the case of evaluating N instances of a *constant-size* f on different sets of inputs. Our question in this case is similar in spirit to the notion of *rate* of a channel in information theory. Can we securely realize N copies of f using Boolean circuits of size $O(N)$?

If f admits a *perfectly* secure protocol in the OT-hybrid model, then the protocol necessarily uses a fixed number of bit operations, independent of any security parameter. Combining N instances of such a protocol with the constant-overhead OT from this work, we get a constant-overhead protocol for evaluating N instances of f.

[12] In fact, the question is open even in the simpler special case of zero-knowledge functionalities. A solution for this special case would imply a solution for the general case by applying the GMW compiler [GMW87] to a constant-overhead protocol with semi-honest security.

Theorem 40 (Constant overhead from perfect security). *Let f be a constant-size functionality that can be computed with perfect (malicious) security in the OT-hybrid model. Then, a constant-overhead OT protocol implies a constant-overhead protocol for computing N instances of f.*

Proof. Let π_f be a perfectly secure protocol for f in the OT-hybrid model, and let c be the number of OT invocations made by π_f. Note that since f is independent of the security parameter λ and π_f has perfect security, c must also be independent of λ. Then, N instances of f can be realized using $cN = O(N)$ instances of OT, which using a constant-overhead OT (and standard MPC composition [Can00, Gol09]) can be realized by circuits of size $O(N)$. □

The existence of perfectly secure protocols in the OT-hybrid model is still quite far from understood. There are negative results for functionalities with big inputs (assuming that the protocol's running time must be polynomial in the input length), as well as for constant-size two-sided functionalities delivering outputs to both parties [IKM+13]. The general case of constant-size *one-sided* functionalities is still open, but positive results for natural functionalities appear in the literature.

Early examples include other flavors of OT, including 1-out-of-k bit-OT, its extension to string-OT (of any fixed length) [BCR86], and instances of "Rabin-OT" that correspond to erasure channels with a rational erasure probability [IPS08].

Perfectly secure protocols for a much broader class of one-sided functionalities were recently obtained in [AP21]. This class includes natural functionalities such as small instances of the millionaire's problem, as well as almost all Boolean one-sided functionalities where the party receiving the output has a smaller input domain than the other party.

Realizing a BSC. An even simpler corollary of Theorem 40 is a constant-overhead protocol for securely realizing N instances of a *binary symmetric channel* (BSC). The feasibility and complexity of securely realizing BSC and other channels was studied in several previous works [IKOS09b, KMN22, ANP+22].

Corollary 41 (Constant-overhead BSC). *Suppose there exists a constant-overhead OT protocol with security against malicious parties. Then there exists a constant-overhead protocol for realizing N instances of the $\mathsf{BSC}_{0.25}$ functionality, which takes a bit b from the sender and delivers $b \oplus e$ to the receiver, where $e = 1$ with probability 0.25 and $e = 0$ otherwise.*

Proof. By Theorem 40, it suffices to show that $\mathsf{BSC}_{0.25}$ perfectly reduces to OT. Consider a deterministic functionality f that takes bits b, e_1^S, e_2^S from the sender and bits e_1^R, e_2^R from the receiver, and delivers $b \oplus ((e_1^S \oplus e_1^R) \wedge (e_2^S \oplus e_2^R))$ to the receiver. Let C_f be a Boolean circuit computing f in the natural way. By the abovementioned theorem of [GIP+14], there is a perfectly secure protocol for C_f in the OT-hybrid model with security up to additive attacks. We argue that applying this protocol with randomly chosen inputs $e_1^S, e_2^S, e_1^R, e_2^R$ yields a perfectly secure protocol for $\mathsf{BSC}_{0.25}$ in the OT-hybrid model. Indeed, letting $e = (e_1^S \oplus e_1^R) \wedge (e_2^S \oplus e_2^R)$, it is not hard to see that a single malicious party cannot bias nor learn any information about e by toggling wire values of C_f. Furthermore, toggling b or the output can be trivially simulated in the ideal model. □

6.3 Reducing the Main Open Question to Simpler Questions

Finally, while we leave open the existence of constant-overhead protocols for general Boolean circuits, our result for OT allows us to reduce this question to a question about a special kind of fault-tolerant circuits.

An Algebraic Manipulation Detection (AMD) circuit [GIP+14] for f is a randomized circuit $\widehat{C}$ that computes f while resisting additive attacks in the following sense: the effect of every additive attack on the wires of $\widehat{C}$ can be simulated by an ideal additive attack on the inputs and outputs of f. As before, in the Boolean case an additive attack can toggle an arbitrary subset of the wires. More formally:

Definition 42 (Boolean AMD circuit). *Let $C : \{0,1\}^n \to \{0,1\}^k$ be a (deterministic or randomized) Boolean circuit. We say that a randomized Boolean circuit $\widehat{C} : \{0,1\}^n \to \{0,1\}^k$ is an ϵ-secure AMD implementation of C if the following holds:*

– Completeness. *For all $x \in \{0,1\}^n$, $\widehat{C}(x) \equiv C(x)$.*

– Security against additive attacks. *For any additive attack A, toggling a subset of the wires of $\widehat{C}$, there exist distributions Δ_{in} over $\{0,1\}^n$ and Δ_{out} over $\{0,1\}^k$ such that for every $x \in \{0,1\}^n$ it holds that*

$$SD\left(\widetilde{C}(x), C(x \oplus \Delta_{\mathsf{in}}) \oplus \Delta_{\mathsf{out}}\right) \leq \epsilon,$$

where $\widetilde{C} \leftarrow A(\widehat{C})$ and SD denotes statistical distance.

Boolean AMD circuits are motivated by the goal of obtaining efficient secure computation protocols in the OT-hybrid model. Indeed, applying the semi-honest GMW protocol [GMW87, Gol09] to the AMD circuit $\widehat{C}$, with a suitable encoding to protect the input and output, yields a secure protocol for C [GIP+14, GIW16]. Combined with a constant-overhead OT protocol, this reduces an affirmative answer to the main open question to the design of constant-overhead AMD circuits.

Theorem 43 (Cf. [GIW16], Claim 18). *Suppose that every Boolean circuit C admits a $2^{-\lambda}$-secure AMD implementation $\widehat{C}$ of size $O(|C|) + \mathsf{poly}(\lambda) \cdot |C|^{0.9}$. Then, a constant-overhead OT protocol implies a constant-overhead protocol for general Boolean circuits.*

The main result of [GIW16] is a construction of AMD circuits with polylogarithmic overhead (in $|C|, \lambda$). Whether this can be improved was left open, but the question was further reduced to the design of two kinds of simple protocols in the honest-majority setting: a protocol that only provides semi-honest security (with a constant fraction of corrupted parties) and a protocol that only guarantees the correctness of the output. This should be contrasted to the approach from [IPS08, DIK10], which reduces the question to the design of honest-majority protocols with security against *malicious* parties.

7 Acknowledgements

We thank the anonymous Eurocrypt reviewers for helpful comments.

E. Boyle supported by AFOSR Award FA9550-21-1-0046, ERC Project HSS (852952), and a Google Research Award. G. Couteau supported by the French Agence Nationale de la Recherche (ANR), under grant ANR-20-CE39-0001 (project SCENE), and the France 2030 ANR Project ANR22-PECY-003 SecureCompute. N. Gilboa supported by ISF grant 2951/20, ERC grant 876110, and a grant by the BGU Cyber Center. Y. Ishai supported by ERC Project NTSC (742754), BSF grant 2018393, and ISF grant 2774/20. L. Kohl is funded by NWO Gravitation project QSC. N. Resch supported in part by ERC H2020 grant No.74079 (ALGSTRONGCRYPTO). P. Scholl is supported by the Danish Independent Research Council under project number 0165-00107B (C3PO) and an Aarhus University Research Foundation starting grant.

References

ABR16. Benny Applebaum, Andrej Bogdanov, and Alon Rosen. A dichotomy for local small-bias generators. *Journal of Cryptology*, 29(3):577–596, July 2016.

ADI+17. Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 223–254. Springer, Heidelberg, August 2017.

AFS03. Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A fast provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2003/230, 2003. https://eprint.iacr.org/2003/230.

AG11. Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 403–415. Springer, 2011.

AHI+17. Benny Applebaum, Naama Haramaty, Yuval Ishai, Eyal Kushilevitz, and Vinod Vaikuntanathan. Low-complexity cryptographic hash functions. In Christos H. Papadimitriou, editor, *ITCS 2017*, volume 4266, pages 7:1–7:31, 67, January 2017. LIPIcs.

AIK04. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175. IEEE Computer Society Press, October 2004.

AIK08. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in nc 0. *Computational Complexity*, 17(1):38–69, 2008.

AK19. Benny Applebaum and Eliran Kachlon. Sampling graphs without forbidden subgraphs and unbalanced expanders with negligible error. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 171–179. IEEE Computer Society, 2019.

AL16. Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1087–1100. ACM Press, June 2016.

Ale03. Michael Alekhnovich. More on average case vs approximation complexity. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 298–307. IEEE Computer Society, 2003.

ALSZ17. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions. *Journal of Cryptology*, 30(3):805–858, July 2017.

AM13. Benny Applebaum and Yoni Moses. Locally computable UOWHF with linear shrinkage. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 486–502. Springer, Heidelberg, May 2013.

ANP$^+$22. Pratyush Agarwal, Varun Narayanan, Shreya Pathak, Manoj Prabhakaran, Vinod M. Prabhakaran, and Mohammad Ali Rehan. Secure non-interactive reduction and spectral analysis of correlations. In *EUROCRYPT 2022, Part III*, pages 797–827, 2022.

AP21. Bar Alon and Anat Paskin-Cherniavsky. On perfectly secure 2PC in the OT-hybrid model. *Theor. Comput. Sci.*, 891:166–188, 2021.

App12. Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 805–816. ACM Press, May 2012.

App15. Benny Applebaum. The cryptographic hardness of random local functions – survey. Cryptology ePrint Archive, Report 2015/165, 2015. https://eprint.iacr.org/2015/165.

App16. Benny Applebaum. Cryptographic hardness of random local functions. *Computational complexity*, 25(3):667–722, 2016.

BBDP22. Zvika Brakerski, Pedro Branco, Nico Döttling, and Sihang Pu. Batch-OT with optimal rate. In *EUROCRYPT 2022, Part II*, LNCS, pages 157–186. Springer, Heidelberg, June 2022.

BCG$^+$17a. Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 336–365. Springer, Heidelberg, December 2017.

BCG$^+$17b. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2105–2122. ACM Press, October / November 2017.

BCG$^+$19a. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

BCG$^+$19b. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Heidelberg, August 2019.

BCG$^+$20. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020.

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

BCG$^+$22. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In *CRYPTO 2022, Part II*, LNCS, pages 603–633. Springer, Heidelberg, August 2022.

BCGI18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

BCR86. Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. Information theoretic reductions among disclosure problems. In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 168–173. IEEE Computer Society, 1986.

Bea92. Donald Beaver. Foundations of secure interactive computing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 377–391. Springer, Heidelberg, August 1992.

Bea96. Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 479–488, 1996.

BGI14. Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

BGI15. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Heidelberg, April 2015.

BGI16. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

BIO14. Joshua Baron, Yuval Ishai, and Rafail Ostrovsky. On linear-size pseudorandom generators and hardcore functions. *Theor. Comput. Sci.*, 554:50–63, 2014.

BISW18. Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Quasi-optimal SNARGs via linear multiprover interactive proofs. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 222–255. Springer, Heidelberg, April / May 2018.

BQ09. Andrej Bogdanov and Youming Qiao. On the security of goldreich's one-way function. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 392–405. Springer, 2009.

BSV19. Andrej Bogdanov, Manuel Sabin, and Prashant Nalini Vasudevan. Xor codes and sparse learning parity with noise. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 986–1004. SIAM, 2019.

BW13. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

Can00. Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptol.*, 13(1):143–202, 2000.

CDM$^+$18. Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. On the concrete security of Goldreich's pseudorandom generator. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 96–124. Springer, Heidelberg, December 2018.

CEMT14. James Cook, Omid Etesami, Rachel Miller, and Luca Trevisan. On the one-way function candidate proposed by goldreich. *ACM Transactions on Computation Theory (TOCT)*, 6(3):14, 2014.

CLY22. Lijie Chen, Jiatu Li, and Tianqi Yang. Extremely Efficient Constructions of Hash Functions, with Applications to Hardness Magnification and PRFs. In *CCC 2022*, pages 23:1–23:37, 2022.

CM01. Mary Cryan and Peter Bro Miltersen. On pseudorandom generators in NC0. In *International Symposium on Mathematical Foundations of Computer Science*, pages 272–284. Springer, 2001.

CRR21. Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent vole and oblivious transfer from hardness of decoding structured ldpc codes. In *Annual International Cryptology Conference*, pages 502–534. Springer, 2021.

dCHI$^+$22. Leo de Castro, Carmit Hazay, Yuval Ishai, Vinod Vaikuntanathan, and Muthu Venkitasubramaniam. Asymptotically quasi-optimal cryptography. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, pages 303–334, 2022.

DGI$^+$19. Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2019.

DIK10. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 445–465. Springer, Heidelberg, May / June 2010.

DMR21. Aurélien Dupin, Pierrick Méaux, and Mélissa Rossi. On the algebraic immunity - resiliency trade-off, implications for goldreich's pseudorandom generator. Cryptology ePrint Archive, Report 2021/649, 2021. https://eprint.iacr.org/2021/649.

FLY22. Zhiyuan Fan, Jiatu Li, and Tianqi Yang. The exact complexity of pseudorandom functions and the black-box natural proof barrier for bootstrapping results in computational complexity. In *STOC '22*, pages 962–975, 2022.

GI14. Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, Heidelberg, May 2014.

GIP$^+$14. Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In David B. Shmoys, editor, *46th ACM STOC*, pages 495–504. ACM Press, May / June 2014.

GIW16. Daniel Genkin, Yuval Ishai, and Mor Weiss. Binary AMD circuits from secure multiparty computation. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 336–366. Springer, Heidelberg, October / November 2016.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

Gol00. Oded Goldreich. Candidate one-way functions based on expander graphs. Cryptology ePrint Archive, Report 2000/063, 2000. https://eprint.iacr.org/2000/063.

Gol09. Oded Goldreich. *Foundations of cryptography: volume 2, basic applications.* Cambridge university press, 2009.

HOSS18. Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. TinyKeys: A new approach to efficient multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 3–33. Springer, Heidelberg, August 2018.

IK02. Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 244–256. Springer, 2002.

IKM$^+$13. Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 600–620. Springer, Heidelberg, March 2013.

IKNP03. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

IKO$^+$11. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and Jürg Wullschleger. Constant-rate oblivious transfer from noisy channels. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 667–684. Springer, Heidelberg, August 2011.

IKOS08. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 433–442. ACM Press, May 2008.

IKOS09a. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Extracting correlations. In *50th FOCS*, pages 261–270. IEEE Computer Society Press, October 2009.

IKOS09b. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Extracting correlations. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 261–270. IEEE Computer Society, 2009.

IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008.

JH22. Ron Rothblum Justin Holmgren. Faster sounder succinct arguments and iops. In *Crypto 2022*, 2022.

Kil88. Joe Kilian. Founding cryptography on oblivious transfer. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 20–31. ACM, 1988.

KMN22. Hamidreza Amini Khorasgani, Hemanta K. Maji, and Hai H. Nguyen. Secure non-interactive simulation: Feasibility and rate. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 767–796. Springer, 2022.

KOS15. Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, Heidelberg, August 2015.

KPTZ13. Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

KZCJ07. Jörg Kliewer, Kamil S Zigangirov, and Daniel J Costello Jr. New results on the minimum distance of repeat multiple accumulate codes. In *Proc. 45th Annual Allerton Conf. Commun., Control, and Computing*, 2007.

LV17. Alex Lombardi and Vinod Vaikuntanathan. Limits on the locality of pseudorandom generators and applications to indistinguishability obfuscation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 119–137. Springer, Heidelberg, November 2017.

MST03a. Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On e-biased generators in NC0. In *44th FOCS*, pages 136–145. IEEE Computer Society Press, October 2003.

MST03b. Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On epsilon-biased generators in NC0. In *Annual Symposium on Foundations of Computer Science*, volume 44, pages 136–145. Citeseer, 2003.

OW14. Ryan ODonnell and David Witmer. Goldreich's prg: evidence for near-optimal polynomial stretch. In *Computational Complexity (CCC), 2014 IEEE 29th Conference on*, pages 1–12. IEEE, 2014.

OZ22. Frédérique E. Oggier and Gilles Zémor. Coding constructions for efficient oblivious transfer from noisy channels. *IEEE Trans. Inf. Theory*, 68(4):2719–2734, 2022.

Roy22. Lawrence Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In *CRYPTO 2022, Part I*, LNCS, pages 657–687. Springer, Heidelberg, August 2022.

RR22. Noga Ron-Zewi and Ron D. Rothblum. Proving as fast as computing: succinct arguments with constant prover overhead. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1353–1363. ACM, 2022.

SGRR19. Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019.

Spi95. Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. In Frank Thomson Leighton and Allan Borodin, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 388–397. ACM, 1995.

TZ06. Jean-Pierre Tillich and Gilles Zémor. On the minimum distance of structured ldpc codes with two variable nodes of degree 2 per parity-check equation. In *2006 IEEE International Symposium on Information Theory*, pages 1549–1553. IEEE, 2006.

YGJL22. Jing Yang, Qian Guo, Thomas Johansson, and Michael Lentmaier. Revisiting the concrete security of goldreich's pseudorandom generator. *IEEE Trans. Inf. Theory*, 68(2):1329–1354, 2022.

YWL$^+$20. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1607–1626. ACM Press, November 2020.

A The IKOS Construction

An alternative method for achieving semi-honest Oblivious Transfer (OT) extension with constant multiplicative overhead was put forth by the work of Ishai, Kushilevitz, Ostrovsky, and Sahai [IKOS08] (hereafter "IKOS"), assuming a polynomial-stretch PRG in NC^0. As discussed below, the IKOS approach suffers from attack under malicious sender and does not seem amenable to low-overhead hardening to malicious agents. Within the semi-honest regime, we present a comparison of the concrete efficiency in terms of Boolean operations and bits communicated per resulting OT correlation, as compared to our constructions.

IKOS construction overview. Assume the existence of a polynomial-stretch pseudorandom generator $G : \{0,1\}^\kappa \to \{0,1\}^N$ with constant locality d. Consider the goal of producing N pseudorandom bit-OT correlations. The IKOS construction proceeds as follows.

1. Perform κ instances of pseudorandom *string*-OT, where each instance has string length $L \cdot N/\kappa$, where L is a constant to be determined later. This step can be achieved with $O(N)$ total cost, via a simple "hybrid encryption" technique for extending short string-OT to long string-OT [IKNP03] (see cost analysis below).
2. Consider the desired N pseudorandom OT two-party functionality
$$g(x, (y_{i,0}, y_{i,1})_{i=1}^N) = ((y_{i,\sigma_i})_{i=1}^N, \emptyset),$$
where $x \in \{0,1\}^\kappa$, $(y_{i,0}, y_{i,1}) \in \{0,1\}^2$, and $\sigma_i = G(x)_i$ for each $i \in [N]$. The constant locality of G implies that g possesses a decomposable affine randomized encoding (DARE) over $\mathbb{F}_2$ of total size $O(n)$ [IK02]. The decomposability property implies that each bit of the randomized encoding can be computed as a function of randomness together with a *single* input bit variable, x_j or $y_{i,b}$. (DARE over $\mathbb{F}_2$ can be viewed as an can be viewed an abstraction of a perfectly secure garbled circuit.)
3. The OT sender computes all symbols of the DARE, where for each element depending on an (unknown) input bit x_j it prepares two options: one for $x_j = 0$, and an alternative for $x_j = 1$.
4. The OT sender and receiver convert the κ pseudorandom string OTs from step 1 to chosen-message string OTs, where for each $j \in [\kappa]$:
   - The jth selection bit is taken to be x_j,
   - The jth string message pair is defined by the concatenation of all DARE symbols depending on bit x_j. Each symbol is of length L bits.
5. As a result of the previous step, the OT receiver learns the full DARE of the function g for the correct inputs, and uses this to evaluate its designated output $(y_{i,\sigma_i})_{i=1}^N$, for $\sigma_i = G(x)_i$.

Insecurity of IKOS against malicious parties. Consider the above procedure within a string OT hybrid model; i.e., given access to string OTs generated with security against malicious parties. The attack space of a malicious receiver in the final OT protocol is limited to its choice of x_j selection bits, which does not constitute an attack. In contrast, however, a malicious sender may provide malformed string message pairs, inconsistent with any fixed choice of input bits $(y_{i,0}, y_{i,1})_{i=1}^N$.

It is not clear how to augment the protocol to prevent such attack, without requiring the sender to somehow prove in zero knowledge that its messages are formed properly. Obtaining zero-knowledge proofs for Boolean circuits with constant computational overhead is a major open problem in this area, and even purely heuristic constructions are not known. Alternatively, one could try to apply special-purpose cut-and-choose techniques such as those used for protecting efficient OT extension protocols against malicious parties [KOS15, Roy22]. However, these techniques are inherently tied to string-OTs whose length is proportional to a security parameter, and seem to require (at least) a polylogarithmic computational overhead when adapted to the case of bit-OT.

A.1 Concrete Efficiency of Semi-Honest IKOS

We consider a concrete instantiation of the above framework. Recall the target is generation of N pseudorandom bit-OT correlations.

Choice of local PRG. For the constant-locality PRG, we take the locality-5 Goldreich PRG with XOR-AND predicate P_5, as defined by
$$P_5(x_1, x_2, x_3, x_4, x_5) = x_1x_2 \oplus x_3 \oplus x_4 \oplus x_5.$$

Based on [YGJL22], with this predicate, the PRG plausibly has stretch $\kappa^{1.19}$ when $\kappa \geq 4096$. Larger stretches can be obtained using more complex predicates, but this reduces the efficiency of the construction.

Size of DARE of OT functionality g. For each output $i \in [n]$, the ith function output of g can be expressed as a linear combination $y_{i,0} \oplus (y_{i,0} \oplus y_{i,1})\sigma_i$, which further expands as $y_{i,0} \oplus (y_{i,0} \oplus y_{i,1})(x_{j_1}x_{j_2} \oplus x_{j_3} \oplus x_{j_4} \oplus x_{j_5})$, where each index $j_1, \ldots, j_5$ is dictated by the ith local PRG output predicate. In what follows, we will denote these simply by x_1, x_2, x_3, x_4, x_5 for notational simplicity.

Depending on the value of $(y_{i,0}, y_{i,1}) \in \{0,1\}^2$, each output bit of g corresponds to one of four possible functions in the input x: Either $(x_1x_2 \oplus x_3 \oplus x_4 \oplus x_5)$, $1 \oplus (x_1x_2 \oplus x_3 \oplus x_4 \oplus x_5)$, the constant function 0, or the constant function 1. We give a DARE for each of these 4 cases below:

1) DARE for $(x_1x_2 \oplus x_3 \oplus x_4 \oplus x_5)$ can be constructed with $L = 7$ bits:
$$\begin{pmatrix} x_1 \oplus r_1, & x_2 \oplus r_2, & \multirow{2}{*}{$x_3 \oplus r_5,$} & \multirow{2}{*}{$x_4 \oplus r_6,$} & \multirow{2}{*}{$x_5 \oplus r_3 \oplus r_4 \oplus r_5 \oplus r_6$} \\ x_1r_2 \oplus r_1r_2 \oplus r_3, & x_2r_1 \oplus r_4, & & & \end{pmatrix},$$
where given DARE values $(z_{1,1}, z_{1,2}, z_{2,1}, z_{2,2}, z_3, z_4, z_5)$, reconstruction of the function output is computed as $z_{1,1}z_{2,1} \oplus z_{1,2} \oplus z_{2,2} \oplus z_3 \oplus z_4 \oplus z_5$. Note that 6 of the 7 DARE values are jointly distributed uniformly over the choice of randomness $(r_1, \ldots, r_6)$, and the final value (i.e., z_5) is fully determined given the previous 6 and the corresponding g function output.
2) DARE for $1 \oplus (x_1x_2 \oplus x_3 \oplus x_4 \oplus x_5)$ can be given identically as above, but with the 7th term $x_5 \oplus r_3 \oplus r_4 \oplus r_5 \oplus r_6$ replaced by $x_5 \oplus r_3 \oplus r_4 \oplus r_5 \oplus r_6 \oplus 1$.
3/4) DARE for the constant functions 0 or 1 can be given by 6 random elements (without dependence on x) together with the corresponding 7th element to yield the correct output.

In each of these 4 cases, the resulting 7 DARE elements have identical distributions as for $(x_1x_2 \oplus x_3 \oplus x_4 \oplus x_5)$ above.

Cost of underlying string OTs. The length of the required string OT message pairs is dictated by the size of the DARE representing the outputs of the function g. Each of the N output OT correlations contributes $L = 7$ DARE bit symbols, whose values each depends on one selection bit x_j. For simplicity of analysis, assume the input-dependency graph of the Goldreich local PRG is symmetric, for which each input bit participates in each role of output computations with equal frequency. (In general, these numbers can be within some small bounded difference without loss of generality.) This means that each bit x_j will select between two messages of length $7N/\kappa$.

First consider the cost of generating *pseudorandom* string OTs of the appropriate parameters. Assuming a setup with κ OTs of length λ, this costs:

– Sender: $C_{\mathsf{prg}}(14N)$
– Receiver: $C_{\mathsf{prg}}(7N)$, plus $4N$ gates to compute $G(x)$

where, recall that $C_{\mathsf{prg}}(n)$ measures the cost of producing n pseudorandom bits using some PRG.

Now, consider the cost of converting these pseudorandom string OTs to chosen-message, for the specific DARE messages. We observe that the selection bits x_j may remain pseudorandom. In addition, recall that 6 of 7 DARE bits for each output are pseudorandom. This means that of the 14 total bits (corresponding to the 7 bit values depending on $x_j = 0$ or $x_j = 1$), 6 may remain uncorrected, in turn implicitly defining the random $r_1, \ldots, r_6$. The remaining 8 bit values are then computed and set by reverse-computing the corresponding r_i values as necessary.

We now count the exact number of gates required for the sender to compute its OT messages:

– OT_1: First message defines r_1, second message is $r_1 \oplus 1$ (cost: 1 NOT)
– OT_2: First message defines r_2, second message is $r_2 \oplus 1$ (cost: 1 NOT)
– OT_3: First message defines $r_3 \oplus r_1r_2$, second message is this $\oplus r_2$ (cost: 1 XOR for second message, 1 AND + 1 XOR to get r_3)
– OT_4: first message defines r_4, second message $r_1 \oplus r_4$ (cost: 1 XOR)
– OT_5: first message gives r_5, second message is $r_5 \oplus 1$ (cost: 1 NOT)
– OT_6: first message gives r_6, second message is $r_6 \oplus 1$ (cost: 1 NOT)
– OT_7: set first message to $r_3 \oplus r_4 \oplus r_5 \oplus r_6$, second message to this $\oplus 1$ (cost: 1 NOT + 3 XOR)

For each of OTs 1–6, the sender must compute an additional XOR, to get the correction bit that is sent to the receiver to fix the second message. For OT_7, both messages need to be fixed. This stage costs an additional 8 XORs, plus sending the 8 correction bits to the receiver.

The total cost (for case (1) of the DARE) is: 5 NOT, 1 AND and 13 XOR gates.

For case (2), we get the same plus 1 extra NOT. For case (3) (constant function 0), the cost is: 1 AND + 12 XOR (8 XOR to fix non-random OT messages, 1 AND + 4 XOR to fix the result of the DARE). For case (4), there is 1 additional NOT to invert the result.

Ignoring NOT gates, 2 of the above 4 cases cost 14 gates, while the last 2 cost 13. Since these each occur with probability $1/4$, the average cost is 13.5 gates.

Meanwhile, for each of its N outputs, the receiver uses 1 AND gate plus 5 XOR gates to evaluate the DARE expression.

Combined IKOS costs. Summing up, we get the following overall costs to produce N instances of random bit-OT:

- Setup: κ base OTs of λ-bit strings
- Communication: $8N$ bits from sender to receiver
- Sender computation: $C_{\mathsf{prg}}(14N) + 13.5N$
- Receiver computation: $C_{\mathsf{prg}}(7N) + 10N$

B Analysis of Sparse LPN

The main result of this section is:

Proposition 44. *Let $n, N \in \mathbb{N}$. For constant $d \geq 2$ there exists a constant $K_d > 0$ such that the following holds for every $\gamma > 0$. For sufficiently large n, there exists a matrix $\mathbf{G} \in \{0,1\}^{N\times n}$ with d-sparse rows so that every nonzero vector $\vec{u}$ satisfying $\vec{u}^\top \cdot \mathbf{G} = \vec{0}$ satisfies $\mathsf{HW}(\vec{u}) \geq n^\gamma/K_d$, and furthermore $N \geq n^{(1-\gamma)d/2+\gamma}$.*

Following [MST03b, CRR21], in order to show the existence of a d-sparse matrix $\mathbf{G}$ with good dual distance properties (which thereby implies resistance to linear tests) we show that there is at $[N] \times [n]$ bipartite matrix with left-degree d which has the unique-neighbours property. This is implied by $> d/2$ expansion.

We now make this more precise. Let (V_1, V_2, E) be a bipartite graph with left vertex set $V_2 = [N]$ and right vertex set $V_2 = [n]$. For a subset $S \subseteq V = V_1 \cup V_2$ we denote by $\Gamma(S) = \{u \in V : vu \in E\}$, i.e., the *neighbours* of S. For a vertex $u \in \Gamma(S)$ we call u a *unique-neighbour* of S if $|\Gamma(\{u\}) \cap S| = 1$, i.e., u is incident to exactly one vertex in S.

Now, there is a natural translation between such bipartite graphs and matrices in $\{0,1\}^{N\times n}$ by taking adjacency matrices. Consider the adjacency matrix $\mathbf{G}$ of a graph G and a vector $\vec{u} \in \{0,1\}^N$; we would like to come up with a simple condition implying $\vec{u}^\top\mathbf{G} \neq \vec{0}$. This exactly means that, if $S \subseteq [N]$ is the support of $\vec{u}$, we have that there is an "odd neighbour" of S, i.e., a vertex $v \in \Gamma(S)$ for which $|\Gamma(\{v\}) \cap S|$ is odd. This is certainly implied by $|\Gamma(\{v\}) \cap S| = 1$, i.e., the condition of v being a unique neighbour of S.

So, to guarantee $\mathbf{G}$ has dual distance at least D it suffices to argue that every subset $S \subseteq [N]$ has a unique neighbour in the corresponding graph G. Recall that we are interested in matrices for which every row has d nonzero entries; this is equivalent to the corresponding graph G having left-degree d.

To guarantee that a subset $S \subseteq [N]$ has a unique neighbour, observe that it suffices to show that $|\Gamma(S)| > (d/2)|S|$. Indeed, if there is no unique neighbour of S then every $v \in \Gamma(S)$ is adjacent to at least 2 vertices in S, implies $|\Gamma(S)| \leq (d/2)|S|$. Thus, the existence of a unique neighbour for every subset $S \subseteq [N]$ of size at most D is guaranteed by $(D, d/2)$-expansion: i.e., the property that for all $S \subseteq [N]$ with $|S| \leq D$ we have $|\Gamma(S)| \geq (d/2)|S|$.

Due to the above the discussion, the following lemma implies Proposition 44.

Lemma 45. *Let $n, N \in \mathbb{N}$. For constant $d \geq 2$ there exists a constant $K_d > 0$ such that the following holds for every $\gamma > 0$. For sufficiently large n, there exists a $(D, d/2)$-bipartite expander graph with N left vertices, n right vertices, and left degree d with $N \leq n^{(1-\gamma)d/2+\gamma}$ and $D \leq n^\gamma/K_d$.*

Proof. We proceed via the probabilistic method: we sample a graph and show that it has the desired properties with positive probability. Consider a random bipartite graph with left vertex set $V_1 = [N]$ and right vertex set $V_2 = [n]$ obtained by for each vertex $v \in V_1$ sampling a random subset of d vertices from V_2 and connecting itself to it. Then, for a subset $S \subseteq V_1$ of size i on the left and a subset of size $d * i/2$ on the right we have that the probability that all of the edges leaving S are incident to vertices in T is at most $\left(\frac{di}{2n}\right)^{di}$. Thus, by summing over all the choices for $2 \le i \le D$,[13] the $\binom{N}{i}$ choices for S and the $\binom{n}{id/2}$ choices for T we have that the failure probability is at most

$$\sum_{i=2}^{D} \binom{N}{i} \binom{n}{\frac{di}{2}} \left(\frac{di}{2n}\right)^{di} \le \sum_{i=2}^{D} \left(\frac{eN}{i}\right)^{i} \left(\frac{2en}{di}\right)^{di/2} \left(\frac{di}{2n}\right)^{di}$$
$$\le \sum_{i=2}^{D} \left(K_d \cdot \frac{i}{2n^\gamma}\right)^{i(d/2-1)}$$

where above we have defined the constant $K_d := (e^{d/2+1} d^{d/2})^{d/2-1}$ and used $N \le n^{d/2-\gamma}$ and the equality $\binom{a}{b} \le \left(\frac{ea}{b}\right)^{b}$.

To bound this sum, we use different bounds for different values of i. For $i = 2$ the contribution is $(K_d \cdot \frac{1}{n^\gamma})^{d-2}$. For $i = 3$ the contribution is $(K_d \cdot \frac{3}{2n^\gamma})^{3d/2-3}$. For $i = 4, \ldots, \log^2 n$ each term is at most $(K_d \cdot \frac{\log^2 n}{2n^\gamma})^{2d-4}$, so the total contribution from these terms is at most $\log^2 n \cdot (K_d \cdot \frac{\log^2 n}{2n^\gamma})^{2d-4}$. Finally, for the terms $i = \log^2 n + 1, \ldots, D$ the bound is at most $(1/2)^{(d/2-1)\log^2 n}$, so the total contribution from these terms is at most $D \cdot (1/2)^{(d/2-1)\log^2 n}$. Recalling the choice of $D \le n^\gamma / K_d$, each of these contributions tends to 0 as $n \to \infty$. The lemma follows.

Remark 46. Before concluding this section, we recall that if $N = \Omega(n^2)$ then sparse primal LPN is susceptible to an attack by Arora and Ge [AG11]. Thus, in setting parameters we will ensure $N = o(n^2)$.

C Instantiating the Correlation-Robust Local PRG

Random Local Functions, *i.e.* function $f(x) = P_1(\pi_1(x)|| \cdots || P_m(\pi_m(x))$ where the $\pi_j : \{0,1\}^n \to \{0,1\}^\ell$ are subset projection functions and the P_j are simple predicates over $\{0,1\}^\ell$, were initially introduced by Goldreich in [Gol00] in an attempt to identify the simplest possible one-way function. When $m \gg n$, it has been conjectured that for suitable predicates and if the subsets corresponding to the π_j's form a good expander, random local functions are actually *pseudorandom generators* in NC^0.

The existence of PRGs in NC^0 was first considered by Cryan and Miltersen in [CM01]. Applebaum, Ishai, and Kushilevitz [AIK04, AIK08] showed that pseudorandom generators with linear stretch $m = O(n)$ exist in a complexity class as low as NC^0_4 (the class of constant depth, polysize circuits where each output bit depends on at most 4 input bits), under standard assumptions for the case of PRG with sublinear stretch, and under an Alekhnovich-style LPN assumption for the case of PRG with linear stretch. In the polynomial stretch regime, Mossel, Shpilka, and Trevisan [MST03a] gave a concrete candidate PRG in NC^0, by instantiating a random local function with $d = 5$, and the predicate

$$P_5 : (x_1, x_2, x_3, x_4, x_5) \mapsto x_1 + x_2 + x_3 + x_4 x_5 \; .$$

They proved that this PRG fools $\mathbb{F}_2$-linear distinguishers for a stretch up to $m(n) = n^{1.25-\varepsilon}$ (for an arbitrary small constant ε). This result was later extended to a larger stretch $n^{1.5-\varepsilon}$ in [OW14]. Subsequently, there have been many studies of this PRG and variants with larger stretch [BQ09, App12, OW14, CEMT14, App15, ABR16, AL16, LV17, BCG$^+$17b, CDM$^+$18].

C.1 Properties of the Predicates

Applebaum and Lovett [AL16] put forward the conjecture that for an appropriate choice of the projection functions π_j, a local PRG is secure as soon as the predicates satisfy three properties: (1) high resiliency, (2) high bit-fixing degree, and (3) high rational degree. They formalized this conjecture by proving that predicates satisfying the above properties yield local PRGs that withstand two powerful classes of attacks: $\mathbb{F}_2$-linear attacks and algebraic attacks. Below, we recall the result of Applebaum and Lovett, and prove that all three properties are preserved under shifts. In particular, this implies that instantiating P with a predicate satisfying (1), (2), and (3) also yields a plausible construction of *correlation-robust* PRG, since all predicates $P_i : x \to P(x \oplus \Delta_i)$ satisfy the same properties, and therefore neither linear attacks nor algebraic attacks can contradict the correlation-robustness of the PRG.

Definition 47 (Predicate properties). *For a predicate $P : \{0,1\}^\ell \to \{0,1\}$:*

[13] Note that the $i = 1$ case never fails by construction.

– *P has* resilience k *if it has no nontrivial correlation with any linear combination of less than or equal to k of its input bits. That is,*
$$\mathbb{E}_{z\in\{0,1\}^\ell}[(-1)^{P(z)+\langle z,\gamma\rangle}]=0$$
for all $\gamma\in\{0,1\}^\ell$ with hamming weight $\leq k$.
– *P has r*-bit-fixing degree *e if, taking the minimum over all restrictions of P by fixing r bits, the minimal $\mathbb{F}_2$-degree of a restriction is e.*
– *P has* rational degree *e if there exist $Q,R:\{0,1\}^\ell\to\{0,1\}$ of degrees $\deg_{\mathbb{F}_2}(Q),\deg_{\mathbb{F}_2}(R)\leq e$, where $R\neq 0$, such that $P(z)R(z)+Q(z)=0\ \forall z\in\{0,1\}^\ell$.*

Note the theorems of Applebaum-Lovett are expressed with n copies of the same predicate P; however, the proofs of the corresponding theorems use only the fact that each predicate individually satisfies the relevant properties.

Theorem 48. *[[AL16] (Thm 1.1.2 & 1.4.2)]*

1. *Let P be a predicate with resilience k and r-bit-fixing degree e. Then for any $s>1$ and $\epsilon>0$, if $k,r,e\geq\Omega(s/\epsilon)$ then P is s-pseudorandom against $\mathbb{F}_2$-linear tests.*
2. *Let P be a predicate with rational degree e. Then for any s, if $e>8s+1$ then P is s-pseudorandom against algebraic tests.*

Proposition 49 (Predicate properties preserved under shift). *Consider a Boolean predicate $P:\{0,1\}^\ell\to\{0,1\}$. Then for any $\Delta\in\{0,1\}^\ell$,*

– *If $P(x)$ is k-resilient, then $P(x+\Delta)$ is k-resilient.*
– *If $P(x)$ has r-bit-fixing degree e, then $P(x+\Delta)$ has r-bit-fixing degree e.*
– *If $P(x)$ has rational degree d, then $P(x+\Delta)$ has rational degree d.*

Proof. The k-resilience of $P(x+\Delta)$ follows from the observation that if $z\overset{\$}{\leftarrow}\{0,1\}^\ell$ then also $z+\Delta$ is uniformly random, so
$$\begin{aligned}\mathbb{E}_{z\in\{0,1\}^\ell}[(-1)^{P(z+\Delta)+\langle z,\gamma\rangle}]&=\mathbb{E}_{z\in\{0,1\}^\ell}[(-1)^{P(z+\Delta)+\langle z+\Delta,\gamma\rangle+\langle\Delta,\gamma\rangle}]\\&=(-1)^{\langle\Delta,\gamma\rangle}\mathbb{E}_{z\in\{0,1\}^\ell}[(-1)^{P(z+\Delta)+\langle z+\Delta,\gamma\rangle}]\\&=(-1)^{\langle\Delta,\gamma\rangle}\cdot 0=0.\end{aligned}$$

For the remaining two properties, we simply require the fact that degree of polynomials is preserved under additive shifts of inputs. Indeed, if we can fix e bits of $P(x)$ and then it is equal to a degree r polynomial $Q(x)$, then $P(x+\Delta)$ after fixing these same e bits is equal to $Q(x'+\Delta')$ where x',Δ' is the restriction of x,Δ to these e bits, and $Q(x'+\Delta')$ has degree r. Similarly, if $P(z)R(z)+Q(z)=0$ for all $z\in\{0,1\}^\ell$ then $P(z+\Delta)R(z+\Delta)+Q(z+\Delta)=0$ for all $z\in\{0,1\}^\ell$ and $Q(z+\Delta),R(z+\Delta)$ have degree equal to $\deg(Q),\deg(R)$, respectively.

Concrete Choices of Predicate. In this work, we will explicitly work with two constructions of correlation-robust local PRGs based on the following two predicates. Firstly, the P_5 predicate given above, which plausibly is secure with stretch $n^{1.5-\varepsilon}$. We use this both as a general-purpose PRG (which allows arbitrary stretch when used iteratively), as well as for the correlation-robust PRG in our dual-LPN construction.

Secondly, in our primal-LPN construction, we use the arity 9 predicate $P_9:=\mathsf{XOR\text{-}MAJ}_{4,5}$, defined as
$$\mathsf{XOR\text{-}MAJ}_{4,5}(x_1,\ldots,x_9)=x_1+x_2+x_3+x_4+\mathsf{majority}(x_5,\ldots,x_9)\ ,$$
where the sum is computed modulo 2. This predicate is 4-resilient, has 2-bit fixing degree 2, and has rational degree 3 (see, e.g., [AL16, Corollary 1.2]). By Theorem 48, it follows that it could plausibly achieve stretch $n^{2.5-\varepsilon}$ (see also [DMR21, Theorem 1] and the surrounding discussion). Lastly, we remark that it is possible to compute this predicate with 17 operations.