Oblivious Transfer with Constant Computational Overhead

Elette Boyle[1], Geoffroy Couteau[2][0000−0002−6645−0106], Niv Gilboa[3][0000−0001−7209−3494], Yuval Ishai[4], Lisa Kohl[5], Nicolas Resch[6][0000−0002−5133−5631], and Peter Scholl[7][0000−0002−7937−8422]

[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.

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 [52] (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 [19] with a *local PRG* [46,62,10], 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 [52]. In spite of significant progress on this and related problems, including constant-overhead protocols for *arithmetic* circuits over large fields [8,24], a solution to the above main question is still elusive; see [36,55] 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 [18]. An important example is the random *oblivious transfer* (OT) correlation,[3] in

[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 5), our results apply to most other natural flavors of OT.

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 [48,47]. 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 [41,36]. A natural approach for improvement would be to follow the "GMW-paradigm" [48], 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 [41]. 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 [56,15,65]. 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.

Pseudorandom Correlation Generators. A recent alternative approach to OT generation is via the tool of *pseudorandom correlation generators* (PCG), put forth in [30,25,28]. 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 [28,27,69,40,26], 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.

[4] See the full version for an explicit attack.

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 $\mathsf{PCG.Expand}$ *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 [52]. A well-known candidate is Goldreich's PRG [46], where significant study has gone toward proving resilience against classes of attacks for particular choices of output predicates P_i [20,4,63,38,5,7,12,61,30,39]. 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 the full version 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 [2], 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 [29,40] and made explicit in [26]. Briefly, this means that we need to verify that the distance of a code related to the matrix is not too small.

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 [50,56,15,65] and to PCG-based protocols with sublinear communication [28,27,69,40,26].

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* [43]. 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 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 [34,54], 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 [3].
- *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 [43]. The best known construction of such circuits has polylogarithmic overhead [44].

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) [25,28]. We first recall their approach, and then explain how to achieve constant overhead.

PCGs for sVOLE from LPN [25,28]. 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 $\{0,1\}^N$, $(\mathbf{G}, \mathbf{G} \cdot \vec{s} \oplus \vec{e}) \stackrel{c}{\approx} (\mathbf{G}, \vec{b})$, where $\mathbf{G} \leftarrow \mathbf{C}, \vec{s} \stackrel{\$}{\leftarrow} \{0,1\}^n, \vec{e} \leftarrow \mathcal{D}$ and $\vec{b} \stackrel{\$}{\leftarrow} \{0,1\}^N$. 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 [45,31]. Further, as observed in [66,27] 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 [58,23,33].

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

- Sample $x \xleftarrow{\$} \{0,1\}^\kappa$ as input for P_1.
- Sample $\vec{s} \xleftarrow{\$} \{0,1\}^n$, $\vec{e} \xleftarrow{\$} \mathcal{D}$ 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 [50]. 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[5], 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 [2], 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

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

[5] 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.

where $\mathbf{G}_j$ is the j-th row of $\mathbf{G}$, and $\Pi_j \colon (\{0,1\}^\kappa)^N \to (\{0,1\}^\ell)^N$ 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 $\{\pi_j(x) \cdot e_j\}_{j \in [N]}$, one can achieve a known-index FSS with constant overhead by using the puncturable PRF construction of [58,23,33] together with an observation in [25], 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 [26].

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 [52,13,9,22,36,64,55,42,37], 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 [67] 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 [11]. Alternatively, the failure probability can be eliminated assuming the conjectured existence of explicit unbalanced expanders or similar combinatorial objects; see, e.g., [52,8] 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.[6] 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 [52], 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*

[6] This should be contrasted with a more fine-grained measure of overhead considered in [17,22,36], 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 $N = \lambda$).

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* $\ell \in O(1)$*.*

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* $\pi_1, \ldots, \pi_N \colon \{0,1\}^\kappa \to \{0,1\}^\ell$ *and predicates* $P_1, \ldots, P_N \colon \{0,1\}^\ell \to \{0,1\}$*. For* $\Delta = (\Delta_1, \ldots, \Delta_N) \in \{0,1\}^N$*, we define the* Δ-shift *of* G *as*

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

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* $\lambda \in \mathbb{N}$

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

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 $\Delta = 0$. Further, 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. For simplicity we will rely on the stronger version in the body of the paper. For a formal definition of distributional correlation-robustness, we refer to the full version.

2.3 Pseudorandom Correlation Generators

We recall the notion of pseudorandom correlation generator (PCG) from [28]. 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 8 (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 9 (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:*

$$\{(R'_0, R'_1) \,|(R_0, R_1) \xleftarrow{\$} \mathcal{C}(1^\lambda), R'_\sigma := R_\sigma, R'_{1-\sigma} \xleftarrow{\$} \mathsf{RSample}(\sigma, R_\sigma)\}$$

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

The following definition of pseudorandom correlation generators is taken almost verbatim from [28]; it generalizes an earlier definition of pseudorandom VOLE generator in [25].

Definition 10 (Pseudorandom Correlation Generator (PCG) [28]). *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:*

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

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

2.4 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 11 (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*

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

Definition 12 (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*

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

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 13 (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 [16], has been further analysed in [25] and [49], and has found applications in the PCG line of work as it significantly improves efficiency [25,28,27]. 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. For more details we refer to the full version.

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 [26],[7] 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 $\mathsf{bias}_{\vec{v}}(\mathsf{Reg}_t^N) \leq (1 - 2(D/t)/(N/t))^t < e^{-2tD/N}$.

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 $\mathsf{HW}(\vec{u}) \geq D$ (say). To see this, note that if $\vec{u}^\top \cdot \mathbf{G} \neq \vec{0}^\top$ 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 $\mathsf{HW}(\vec{u}) \geq D$ and $\mathsf{HW}(\vec{e}) \geq t$. 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 $\mathsf{HW}(\vec{u}^\top \cdot \mathbf{H}) \geq D$ 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 $2^{-\lambda}$.

2.5 (Known Index) 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 [32] by allowing some additional leakage given to one of the parties. The leakage will be the point/ interval positions to party P_0. As observed in [66,27], 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 [32], 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 output domain 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 $\mathbb{G}$.

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

Note that as a difference to the original definition, we include $\mathsf{FullEval}$ in the full 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 14 (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 15 (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 14, 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 16 (Pseudorandomness of the output shares). In [31] 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 17 (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 18 (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 regular multi-point functions and projected-payload multi-interval function. For more definitions we refer to the full version.

Definition 19 (Regular multi-point function). *Let* $t \in \mathbb{N}, N \in \mathbb{N}$*,* $\alpha_1, \ldots, \alpha_t \in [N/t]$*,* $\mathbb{G}$ *be an additive group and* $\beta_1, \ldots, \beta_t \in \mathbb{G}$*. The* regular multi-point function *defined by* $\vec{\alpha} = (\alpha_1, \ldots, \alpha_t)$ *and* $\vec{\beta} = (\beta_1, \ldots, \beta_t)$ *is then*

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

Definition 20 (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* $\alpha_1, \ldots, \alpha_t \in [N]$ *be pairwise different and* $\beta \in \{0,1\}^\kappa$*. Then, we define the projected-payload interval function specified by* $\vec{\alpha} = (\alpha_1, \ldots, \alpha_t)$*,* β *and* $\vec{\pi} := (\pi_1, \ldots, \pi_N)$ *as*

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

Known-index FSS. In known-index FSS for point functions, introduced in [66], the index α is allowed to be leaked to party P_0. As observed in [66,27], 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 [26], 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 full version we give the formal definitions of these flavors of FSS, present constructions and analyze their circuit complexity. For known-index DPF and known-index interval FSS, the constructions are based on prior works, while for projected-payload FSS, we devise a new construction.

3 Constant-Overhead PCG for OT from Primal LPN

In this section give a PCG for OT with constant overhead in Figure 1. An inherent limitation to this approach is that primal LPN is limited to subquadratic stretch.

First, following Alekhnovich [2], 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 $N = \Omega(n^{d/2})$ [6,21]: 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 2.4 when the dual distance D is larger we obtain security against linear tests: the security is at most $2^{-\lambda}$ 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 [14], 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 [11] 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 21. *Let $N = N(\lambda), n = n(\lambda), t = t(\lambda), \kappa = \kappa(\lambda) \in \mathbb{N}$ and let $\ell, d \in \mathbb{N}$ be constant and $\mathbf{C}$ a primal code generation algorithm with constant 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 1 is a PCG for generating N instances of the bit-OT correlation.*

The proof is provided in the full version.

Lemma 22. *The* $\mathsf{PCG.Gen}$ *algorithm in Fig. 1 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.

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 $\mathbf{G} \xleftarrow{\$} \mathbf{C}(N, n, \mathbb{F}_2)$ 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* $x \xleftarrow{\$} \{0,1\}^\kappa$ at random.
- Pick an *LPN seed* $\vec{s} \xleftarrow{\$} \{0,1\}^n$ 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 $\vec{r}_1 \xleftarrow{\$} (\{0,1\}^\kappa)^n$ and set $\vec{r}_1 := \vec{r}_0 \oplus \vec{r}$.
- Choose *regular noise positions* $\vec{\alpha} \xleftarrow{\$} [N/t]^t$ 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 $\pi_j(\vec{y}_j^0) \in \{0,1\}^\ell$ 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 $w_j := P_j(\pi_j(y_j^0) \oplus v_j^0) \in \{0,1\}$.
   - 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 $w_{j,b} := P_j(\pi_j(y_j^1) \oplus \vec{v}_j^1 \oplus b \cdot \pi_j(\vec{x}))$.
   - Output $\{(w_{j,0}, w_{j,1})\}_{j \in [N]}$.

Fig. 1: Constant-overhead PCG for N instances of random bit-OT.

- 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 $w_j / \ w_{j,0}, w_{j,1}$ can be computed with $C_P + 1$ gates, resulting in either $(C_P + 1)N$ (for $\sigma = 0$) or $2(C_P + 1)N$ ($\sigma = 1$) for the last step.

We can instantiate the FSS construction with the naive "square-root" construction of known-index DPF from the full version. 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 23. *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 [54], 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 22 and plugging in the square-root FSS construction, we obtain a protocol that securely computes the $\mathsf{PCG.Gen}$ algorithm from Fig. 1 with complexity $(\kappa n + \lambda\sqrt{Nt}) \cdot p(\lambda)$. Following [28, 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 22, 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 $\mathrm{XOR}_4 \oplus \mathrm{MAJ}_5$ predicate 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 [14]. 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 24 (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 25 (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 [25].

4 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. All proofs in this section are provided in the full version.

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

Definition 26. *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 $\vec{e} \xleftarrow{\$} \mathsf{Reg}_t^M(\{0,1\})$, 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 [60, 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.[8]

Theorem 27. *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 $(\mathsf{Reg}_t^M(\{0,1\}), \mathbf{H}, \{0,1\})$-$\mathsf{LPN}(N, M)$-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 2 is a constant-overhead PCG for generating N instances of the bit-OT correlation.*

Lemma 28. *The $\mathsf{PCG.Gen}$ algorithm in Fig. 2 has circuit size $O(t \log(N/t) + C_{\mathsf{FSS.Setup}}(\lambda))$. 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*

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

Corollary 29. *Suppose OT exists. Suppose the $(\mathsf{Reg}_t^M(\mathbb{F}_2), \mathbf{H}, \mathbb{F}_2)$-$\mathsf{LPN}(N, M)$-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.

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).

5 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

[8] Instead of RA codes we could have used a code of Tillich and Zémor [68]; however the effect on the computational complexity is essentially nil.

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 $(\mathsf{Reg}_t^M(\{0,1\}), \mathbf{H}, \{0,1\})$-$\mathsf{LPN}(N, dN)$ 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. 2: Constant-overhead PCG for N instances of random bit-OT.

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 [52], extending this to the malicious model was posed as an open question.[9] The overhead of the best known protocols grows polylogarithmically with the security parameter and the circuit size [41].

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

5.1 General Protocols with Relaxed Security

Our main result gives the first constant-overhead implementation of (malicious bit-) OT. However, extending this to general functionalities is challenging. While it is well-known that OT is complete for secure computation [59,54], the best known protocols for Boolean circuits in the OT-hybrid model have polylogarithmic overhead [41,44]. In contrast, in the *semi-honest* OT-hybrid model, a simple "textbook" protocol [48,47], commonly referred to as the (semi-honest) *GMW protocol*, achieves perfect security with a small constant overhead.

A key observation from [43] 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 [35,47], we get the following application of our main result.

Theorem 30 (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 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

[9] 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 [48] to a constant-overhead protocol with semi-honest security.

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 [52] to such a functionality may allow a malicious party to learn the entire input of the honest party.

5.2 Leveraging Perfect Security

Consider the case of evaluating N instances of a constant-size f on different sets of inputs. 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.

Theorem 31 (Constant overhead from perfect security). *Let f be a constant-size functionality that can be computed with perfect security in the OT-hybrid model. Then, a constant-overhead OT protocol with security against malicious parties implies a constant-overhead protocol for computing N instances of f.*

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 [51]. 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) [34], and instances of "Rabin-OT" that correspond to erasure channels with a rational erasure probability [54].

Perfectly secure protocols for a much broader class of one-sided functionalities were recently obtained in [3]. 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 31 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 [53,57,1].

Corollary 32 (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 31, 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 [43], 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.

5.3 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 [43] 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 33 (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 [48,47] to the AMD circuit $\widehat{C}$, with a suitable encoding to protect the input and output, yields a secure protocol for C [43,44]. 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 34 (Cf. [44], 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 [44] 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 [54,41], which reduces the question to the design of honest-majority protocols with security against *malicious* parties.

6 Acknowledgements

E. Boyle supported by AFOSR Award FA9550-21-1-0046, ERC Project HSS (852952), and a Google Research Award. G. Couteau supported by the ANR SCENE. 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

1. Agarwal, P., Narayanan, V., Pathak, S., Prabhakaran, M., Prabhakaran, V.M., Rehan, M.A.: Secure non-interactive reduction and spectral analysis of correlations. In: Dunkelman, O., Dziembowski, S. (eds.) 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. Lecture Notes in Computer Science, vol. 13277, pp. 797–827. Springer (2022). https://doi.org/10.1007/978-3-031-07082-2_28, https://doi.org/10.1007/978-3-031-07082-2_28
2. Alekhnovich, M.: 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. pp. 298–307. IEEE Computer Society (2003). https://doi.org/10.1109/SFCS.2003.1238204, https://doi.org/10.1109/SFCS.2003.1238204
3. Alon, B., Paskin-Cherniavsky, A.: On perfectly secure 2PC in the OT-hybrid model. Theor. Comput. Sci. **891**, 166–188 (2021). https://doi.org/10.1016/j.tcs.2021.08.035, https://doi.org/10.1016/j.tcs.2021.08.035
4. Applebaum, B.: Pseudorandom generators with long stretch and low locality from random local one-way functions. In: 44th ACM STOC (May 2012)

5. Applebaum, B.: The cryptographic hardness of random local functions – survey. Cryptology ePrint Archive, Report 2015/165 (2015), https://eprint.iacr.org/2015/165
6. Applebaum, B.: Cryptographic hardness of random local functions. Computational complexity **25**(3), 667–722 (2016)
7. Applebaum, B., Bogdanov, A., Rosen, A.: A dichotomy for local small-bias generators. Journal of Cryptology (3) (Jul 2016)
8. Applebaum, B., Damgård, I., Ishai, Y., Nielsen, M., Zichron, L.: Secure arithmetic computation with constant computational overhead. In: CRYPTO 2017, Part I (Aug 2017)
9. Applebaum, B., Haramaty, N., Ishai, Y., Kushilevitz, E., Vaikuntanathan, V.: Low-complexity cryptographic hash functions. In: ITCS 2017 (Jan 2017)
10. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in NC^0. In: 45th FOCS (Oct 2004)
11. Applebaum, B., Kachlon, E.: Sampling graphs without forbidden subgraphs and unbalanced expanders with negligible error. In: Zuckerman, D. (ed.) 60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019. pp. 171–179. IEEE Computer Society (2019). https://doi.org/10.1109/FOCS.2019.00020, https://doi.org/10.1109/FOCS.2019.00020
12. Applebaum, B., Lovett, S.: Algebraic attacks against random local functions and their countermeasures. In: 48th ACM STOC (Jun 2016)
13. Applebaum, B., Moses, Y.: Locally computable UOWHF with linear shrinkage. In: EUROCRYPT 2013 (May 2013)
14. Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I. Lecture Notes in Computer Science, vol. 6755, pp. 403–415. Springer (2011)
15. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer extensions. Journal of Cryptology (3) (Jul 2017)
16. Augot, D., Finiasz, M., Sendrier, N.: A fast provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2003/230 (2003), https://eprint.iacr.org/2003/230
17. Baron, J., Ishai, Y., Ostrovsky, R.: On linear-size pseudorandom generators and hardcore functions. Theor. Comput. Sci. **554**, 50–63 (2014). https://doi.org/10.1016/j.tcs.2014.06.013, https://doi.org/10.1016/j.tcs.2014.06.013
18. Beaver, D.: Foundations of secure interactive computing. In: CRYPTO'91 (Aug 1992)
19. Beaver, D.: Correlated pseudorandomness and the complexity of private computations. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. pp. 479–488 (1996)
20. Bogdanov, A., Qiao, Y.: On the security of goldreich's one-way function. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pp. 392–405. Springer (2009)
21. Bogdanov, A., Sabin, M., Vasudevan, P.N.: Xor codes and sparse learning parity with noise. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 986–1004. SIAM (2019)
22. Boneh, D., Ishai, Y., Sahai, A., Wu, D.J.: Quasi-optimal SNARGs via linear multi-prover interactive proofs. In: EUROCRYPT 2018, Part III (Apr / May 2018)

23. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: ASIACRYPT 2013, Part II (Dec 2013)
24. Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: ASIACRYPT 2017, Part III (Dec 2017)
25. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: ACM CCS 2018 (Oct 2018)
26. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Resch, N., Scholl, P.: Correlated pseudorandomness from expand-accumulate codes. In: Advances in Cryptology - CRYPTO 2022 (2022), https://eprint.iacr.org/2022/1014
27. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: ACM CCS 2019 (Nov 2019)
28. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: CRYPTO 2019, Part III (Aug 2019)
29. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Correlated pseudorandom functions from variable-density LPN. In: 61st FOCS (Nov 2020)
30. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Orrù, M.: Homomorphic secret sharing: Optimizations and applications. In: ACM CCS 2017 (Oct / Nov 2017)
31. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: EUROCRYPT 2015, Part II (Apr 2015)
32. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: ACM CCS 2016 (Oct 2016)
33. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: PKC 2014 (Mar 2014)
34. Brassard, G., Crépeau, C., Robert, J.: Information theoretic reductions among disclosure problems. In: 27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986. pp. 168–173. IEEE Computer Society (1986). https://doi.org/10.1109/SFCS.1986.26, https://doi.org/10.1109/SFCS.1986.26
35. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000). https://doi.org/10.1007/s001459910006, https://doi.org/10.1007/s001459910006
36. de Castro, L., Hazay, C., Ishai, Y., Vaikuntanathan, V., Venkitasubramaniam, M.: Asymptotically quasi-optimal cryptography. In: Dunkelman, O., Dziembowski, S. (eds.) 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 I. Lecture Notes in Computer Science, vol. 13275, pp. 303–334. Springer (2022). https://doi.org/10.1007/978-3-031-06944-4_11, https://doi.org/10.1007/978-3-031-06944-4_11
37. Chen, L., Li, J., Yang, T.: Extremely Efficient Constructions of Hash Functions, with Applications to Hardness Magnification and PRFs. In: Lovett, S. (ed.) 37th Computational Complexity Conference (CCC 2022). Leibniz International Proceedings in Informatics (LIPIcs), vol. 234, pp. 23:1–23:37. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2022). https://doi.org/10.4230/LIPIcs.CCC.2022.23, https://drops.dagstuhl.de/opus/volltexte/2022/16585

38. Cook, J., Etesami, O., Miller, R., Trevisan, L.: On the one-way function candidate proposed by goldreich. ACM Transactions on Computation Theory (TOCT) **6**(3), 14 (2014)
39. Couteau, G., Dupin, A., Méaux, P., Rossi, M., Rotella, Y.: On the concrete security of Goldreich's pseudorandom generator. In: ASIACRYPT 2018, Part II (Dec 2018)
40. Couteau, G., Rindal, P., Raghuraman, S.: Silver: Silent vole and oblivious transfer from hardness of decoding structured ldpc codes. In: Annual International Cryptology Conference. pp. 502–534. Springer (2021)
41. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly secure multiparty computation and the computational overhead of cryptography. In: EUROCRYPT 2010 (May / Jun 2010)
42. Fan, Z., Li, J., Yang, T.: The exact complexity of pseudorandom functions and the black-box natural proof barrier for bootstrapping results in computational complexity. In: Leonardi, S., Gupta, A. (eds.) STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022. pp. 962–975. ACM (2022). https://doi.org/10.1145/3519935.3520010, `https://doi.org/10.1145/3519935.3520010`
43. Genkin, D., Ishai, Y., Prabhakaran, M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: 46th ACM STOC (May / Jun 2014)
44. Genkin, D., Ishai, Y., Weiss, M.: Binary AMD circuits from secure multiparty computation. In: TCC 2016-B, Part I (Oct / Nov 2016)
45. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: EUROCRYPT 2014 (May 2014)
46. Goldreich, O.: Candidate one-way functions based on expander graphs. Cryptology ePrint Archive, Report 2000/063 (2000), `https://eprint.iacr.org/2000/063`
47. Goldreich, O.: Foundations of cryptography: volume 2, basic applications. Cambridge university press (2009)
48. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A.V. (ed.) Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA. pp. 218–229. ACM (1987). https://doi.org/10.1145/28395.28420, `https://doi.org/10.1145/28395.28420`
49. Hazay, C., Orsini, E., Scholl, P., Soria-Vazquez, E.: TinyKeys: A new approach to efficient multi-party computation. In: CRYPTO 2018, Part III (Aug 2018)
50. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: CRYPTO 2003 (Aug 2003)
51. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: TCC 2013 (Mar 2013)
52. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with constant computational overhead. In: 40th ACM STOC (May 2008)
53. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Extracting correlations. In: 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA. pp. 261–270. IEEE Computer Society (2009). https://doi.org/10.1109/FOCS.2009.56, `https://doi.org/10.1109/FOCS.2009.56`
54. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: CRYPTO 2008 (Aug 2008)
55. Justin Holmgren, R.R.: Faster sounder succinct arguments and iops. In: Crypto 2022 (2022)

56. Keller, M., Orsini, E., Scholl, P.: Actively secure OT extension with optimal overhead. In: CRYPTO 2015, Part I (Aug 2015)
57. Khorasgani, H.A., Maji, H.K., Nguyen, H.H.: Secure non-interactive simulation: Feasibility and rate. In: Dunkelman, O., Dziembowski, S. (eds.) 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. Lecture Notes in Computer Science, vol. 13277, pp. 767–796. Springer (2022). https://doi.org/10.1007/978-3-031-07082-2_27, https://doi.org/10.1007/978-3-031-07082-2_27
58. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: ACM CCS 2013 (Nov 2013)
59. Kilian, J.: Founding cryptography on oblivious transfer. In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA. pp. 20–31. ACM (1988). https://doi.org/10.1145/62212.62215, https://doi.org/10.1145/62212.62215
60. Kliewer, J., Zigangirov, K.S., Costello Jr, D.J.: New results on the minimum distance of repeat multiple accumulate codes. In: Proc. 45th Annual Allerton Conf. Commun., Control, and Computing (2007)
61. Lombardi, A., Vaikuntanathan, V.: Limits on the locality of pseudorandom generators and applications to indistinguishability obfuscation. In: TCC 2017, Part I (Nov 2017)
62. Mossel, E., Shpilka, A., Trevisan, L.: On e-biased generators in NC0. In: 44th FOCS (Oct 2003)
63. ODonnell, R., Witmer, D.: Goldreich's prg: evidence for near-optimal polynomial stretch. In: Computational Complexity (CCC), 2014 IEEE 29th Conference on. pp. 1–12. IEEE (2014)
64. Ron-Zewi, N., Rothblum, R.D.: Proving as fast as computing: succinct arguments with constant prover overhead. In: Leonardi, S., Gupta, A. (eds.) STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022. pp. 1353–1363. ACM (2022). https://doi.org/10.1145/3519935.3519956, https://doi.org/10.1145/3519935.3519956
65. Roy, L.: Softspokenot: Communication-computation tradeoffs in OT extension. In: Crypto 2022 (2022)
66. Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed vector-OLE: Improved constructions and implementation. In: ACM CCS 2019 (Nov 2019)
67. Spielman, D.A.: Linear-time encodable and decodable error-correcting codes. In: Leighton, F.T., Borodin, A. (eds.) Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA. pp. 388–397. ACM (1995). https://doi.org/10.1145/225058.225165, https://doi.org/10.1145/225058.225165
68. Tillich, J.P., Zémor, G.: 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. pp. 1549–1553. IEEE (2006)
69. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated OT with small communication. In: ACM CCS 2020 (Nov 2020)