Sublinear-Communication Secure Multiparty Computation Does Not Require FHE

Elette Boyle[1,2], Geoffroy Couteau[3], and Pierre Meyer[1,3(✉)]

[1] Reichman University, Herzliya, Israel
eboyle@alum.mit.edu
[2] NTT Research, Sunnyvale, USA
[3] Université Paris Cité, CNRS, IRIF, Paris, France
{couteau,pierre.meyer}@irif.fr

Abstract. Secure computation enables mutually distrusting parties to jointly compute a function on their secret inputs, while revealing nothing beyond the function output. A long-running challenge is understanding the required communication complexity of such protocols—in particular, when communication can be *sublinear* in the circuit representation size of the desired function. Significant advances have been made affirmatively answering this question within the *two-party* setting, based on a variety of structures and hardness assumptions. In contrast, in the *multi-party* setting, only one general approach is known: using Fully Homomorphic Encryption (FHE). This remains the state of affairs even for just three parties, with two corruptions.

We present a framework for achieving secure sublinear-communication $(N+1)$-party computation, building from a particular form of Function Secret Sharing for only N parties. In turn, we demonstrate implications to sublinear secure computation for various function classes in the 3-party and 5-party settings based on an assortment of assumptions not known to imply FHE.

Keywords: Foundations · Secure Multiparty Computation · Function Secret Sharing · Private Information Retrieval

1 Introduction

Secure computation enables mutually distrusting parties to jointly compute a function on their secret inputs, while revealing nothing beyond the function output. Since the seminal feasibility results of the 1980s [6,18,29,41], a major challenge in the area has been if and when it is possible to break the "circuit-size barrier." This barrier refers to the fact that all classical techniques for secure computation required a larger amount of communication than the size of a boolean

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-30617-4_6.

C. Hazay and M. Stam (Eds.): EUROCRYPT 2023, LNCS 14005, pp. 159–189, 2023.
https://doi.org/10.1007/978-3-031-30617-4_6

circuit representing the function to be computed. In contrast, insecure computation only requires exchanging the inputs, which are usually considerably smaller than the entire circuit.

This challenge eluded the field for nearly two decades, aside from partial results that either required exponential computation [4,36], or were limited to very simple functions (such as point functions [20,21,35] or constant-depth circuits [3]). This changed with the breakthrough result of Gentry [27] on *fully homomorphic encryption* (FHE). FHE is a powerful primitive supporting computation on encrypted data, which can be used to build asymptotically optimal-communication protocols in the computational setting [2,24].

In the years after, significant progress has been made toward broadening the set of techniques and class of assumptions under which sublinear-communication secure computation can be built. A notable such approach is via *homomorphic secret sharing* (HSS) [11]. HSS can be viewed as a relaxation of FHE, where homomorphic evaluation can be distributed among two parties who do not interact with each other, but which still suffices for low-communication secure computation. Following this approach (explicitly, building forms of HSS for NC^1), sublinear-communication secure protocols have been developed based on the Decisional Diffie-Hellman (DDH) assumption [11], Decision Composite Residuosity (DCR) [26,37,40], and further algebraic structures, including a class of assumptions based on class groups of imaginary quadratic fields [1]. It was extended to a flavor of the Learning Parity with Noise (LPN) assumption (via HSS for log log-depth circuits) by [23]. Othogonally to these approaches, which rely on computational assumptions, [22] built sublinear-communication secure computation under an assumption of correlated randomness.

Very recently, a work of [9] demonstrated an alternative approach to sublinear secure computation through a certain form of rate-1 batch oblivious transfer (OT), resulting in protocols based on a weaker form of LPN plus Quadratic Residuosity.

However, aside from the original approach via FHE, *all* of the above techniques are strongly tied to the *two-party* setting, as opposed to the general setting of multiple parties, where all but one can be corrupt.

More concretely, while N-party HSS with security against $(N-1)$ colluding parties would directly imply the desired result, actually achieving such a primitive for rich function classes (without tools already implying FHE) beyond $N = 2$, is a notable open challenge in the field. The 2-party setting provides special properties leveraged within HSS constructions; e.g., given an additive secret sharing of 0, it implies the two parties hold identical values. These properties completely break down as soon as one steps to three parties with security against two. This separation can already be showcased for very simple function classes, such as HSS for equality test (equivalently, "distributed point functions" [10,28]), where to this date an exponential gap remains between the best constructions in the 2-party versus 3-party setting [10]. For $N \geq 3$, there are constructions of N-party FSS for all polynomial-time computable functions, but only from LWE, by using *additive-function-sharing* spooky encryption (AFS-spooky encryption) [25], or from subexponentially secure *indistinguishability obfuscation* [10].

Additionally, [14] turns this FSS from spooky encryption into additive HSS. In addition, approaches from the 2-party batch OT primitive seem also to be strongly tied to two parties.

Despite great progress in the two-party setting—and the fundamental nature of the question—to date, sublinear secure computation results for 3 or more parties remain stuck in the "2009 era": known only for very simple functions (e.g., constant-degree computations), or based on (leveled) FHE.

1.1 Our Results

We present a new framework for achieving secure computation with low communication. Ultimately our approach yields new sublinear secure computation protocols for various circuit classes in the 3-party and 5-party settings, based on an assortment of assumptions not known to imply FHE.

General Framework. Our high-level approach centers around Function Secret Sharing (FSS) [10], a form of secret sharing where the secret object and shares comprise succinct representations of *functions*. More concretely, FSS for function class $\mathcal{F}$ allows a client to split a secret function $f \in \mathcal{F}$ into function shares $f_1, \ldots, f_N$ such that any strict subset of f_i's hide f, and for every input x in the domain of f it holds that $\sum_{i=1}^{N} f_i(x) = f(x)$. (This can be seen as the syntactic dual of HSS, where the role of input and function are reversed; we refer the reader to e.g. [14] for discussion.[1]) N-party FSS/HSS for sufficiently rich function classes is known to support low-communication N-party secure computation, but lack of multi-party FSS constructions effectively leaves us stuck at $N = 2$.

The core conceptual contribution of this work is the following simple framework, which enables us to achieve $(N+1)$-party secure computation by using a form of FSS for only N parties.

Proposition 1 ($(N+1)$-PC from N-FSS framework, informal). *For any ensemble of polynomial-size circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}$, consider an N-party FSS scheme for the class of "partial evaluation" functions $\{C_\lambda(\cdot, x_1, \ldots, x_N)\}_{\lambda, x_1, \ldots, x_N}$, and define the following sub-computation functionalities:*

- $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$*: N-party secure FSS share distribution, where each party P_i holds input x_i (and λ), and learns the ith FSS key f_i for the function $C_\lambda(\cdot, x_1, \ldots, x_N)$.*
- $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$*: Two-party oblivious FSS evaluation, where party P_i holds an FSS key f_i, party P_0 holds input x_0, and P_0 learns the ith output $f_i(x_0)$.*

Then there exists a $(N+1)$-party protocol for securely computing $\mathcal{C}$ making one call to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ and N calls to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$.

Once expressed in this form, the resulting $(N+1)$-party protocol becomes an exercise: Roughly, it begins by having parties $1, \ldots, N$ jointly execute $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ on their inputs $x_1, \ldots, x_N$ to each receive a function share f_i of the secret function $f(x_0) := C_\lambda(x_0, x_1, \ldots, x_N)$, and then each run a pairwise execution of $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$

[1] Indeed, we will refer to both notions, using each when more conceptually convenient.

together with the remaining party P_0 in order to obliviously communicate the ith output share $f_i(x)$. Given these shares, P_0 can compute the final output as $\sum_{i=1}^{N} f_i(x_0)$. (See the Technical Overview for more detailed discussion.)

The communication of the resulting protocol will be dominated by the executions of $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$. Of course, the technical challenge thus becomes if and how one can construct corresponding FSS schemes which admit secure share distribution and oblivious evaluation with *low communication*.

Instantiating the Framework. We demonstrate how to instantiate the above framework building from known constructions of Homomorphic Secret Sharing (HSS) combined with a version of low-communication PIR.

We first identify a structural property of an FSS scheme which, if satisfied, then yields a low-communication procedure for oblivious share evaluation, through use of a certain notion of "correlated" (batch) Symmetric Private Information Retrieval (SPIR). Loosely, correlated SPIR corresponds to a primitive where a client wishes to make *correlated* queries into m distinct size-S databases held by a single server. Without correlation between queries, the best-known PIR constructions would require $m \cdot \mathsf{polylog}(S)$ communication. However, it was shown in [9] that if the m index queries (each $\log S$ bits) are given by various subsets of a fixed bit string of length $n \ll m \log S$ held by the client, then (using the rate-1 batch OT constructions from [17]) this batch SPIR can be performed with significantly lower communication.

We then demonstrate that FSS schemes with the necessary structural property can be realized from existing constructions of HSS. Loosely speaking, the FSS evaluation procedure will be expressible as a polynomial (which depends on $x_1, \ldots, x_N$) evaluated on the final input x_0, and the HSS will enable the N parties to compute additive secret shares of the coefficients of this corresponding polynomial.

We further extend the approach to support an underlying HSS scheme satisfying only a weaker notion of correctness, with *inverse-polynomial* (Las Vegas) *error*. In such scheme, homomorphic evaluation may fail with noticeable probability (over the randomness of share generation), in a manner identifiable to one or more parties. This is the notion satisfied by the 2-party HSS constructions from Decisional Diffie-Hellman [11], or Learning With Errors with only a polynomial-size modulus [16,25]. This error must be removed in our construction while incurring minimal additional interaction. We demonstrate how to do so, using (standard) Private Information Retrieval [21] and punctured pseudorandom functions [8,15,33]. Note that the former is implied by correlated SPIR, and the latter implied by any one-way function, so that these tools do not impose additional assumptions in the statement below.

Theorem 2 (Sublinear MPC, informal). *For any ensemble of polynomial-size circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}$ of size s, depth $\log \log s$, and with n inputs and m outputs, if there exists the following:*

- *Correlated Symmetric Batch PIR, for m size-s databases where queries come from n bits, with communication $O(n + m + \mathsf{poly}(\lambda) + \mathsf{comm}(s))$ for some function* comm.

– *(Las Vegas) N-party Homomorphic Secret Sharing with compact shares (size $O(n)$ for input size n), for the class of* $\log\log$*-depth boolean circuits.*

Then there exists a secure $(N+1)$-party computation protocol for $\mathcal{C}$ with communication $O(n+m+\mathsf{poly}(\lambda)+N\cdot\mathsf{comm}(s))$. In particular, sublinearity is achieved when $N\cdot\mathsf{comm}(s)\in o(s)$.

Remark 3 (Compiling Sublinear MPC from Passive to Active Security). In this work, we focus on security against semi-honest adversaries. However, all our results extend immediately to the malicious setting, using known techniques. Indeed, to get malicious security while preserving sublinearity, one can just use the seminal GMW compiler [29] with zero-knowledge arguments, instantiating the ZKA with (interactive) succinct arguments [36]. Using Kilian's PCP-based 4-move argument [34], which has polylogarithmic communication, this can be done using any collision-resistant hash function. The latter are implied by all assumptions under which we base sublinear MPC, hence our results generalise directly to the malicious setting. This observation was made in previous works on sublinear-communication secure computation (e.g. [9,11,23]).

Remark 4 (Beyond Boolean circuits). The above approach can be extended to arithmetic circuits over general fields $\mathbb{F}$, by replacing the correlated SPIR with an analogous form of (low-communication) *correlated oblivious polynomial evaluation* (OPE). We discuss and prove this more general result in the main body, but focus here on the Boolean setting, as required instantiations of such correlated-OPE beyond constant-size fields are not yet currently known.

Resulting Constructions. Finally, we turn to the literature to identify constructions of the required sub-tools, yielding resulting sublinear secure computation results from various mathematical structures and computational assumptions.

Corollary 5 (Instantiating the framework, informal). *There exists secure 3-party computation for evaluating Boolean circuits of size s and depth* $\log\log s$ *with n inputs and m outputs, with communication complexity $O(n+m+\sqrt{s}\cdot\mathsf{poly}(\lambda)\cdot(n+m)^{2/3})$ based on the Learning Parity with Noise (LPN) assumption for any inverse-polynomial error rate, together with any of the following additional computational assumptions:*

– *Decisional Diffie-Hellman (DDH)*
– *Learning with Errors with polynomial-size modulus (poly-modulus LWE)*
– *Quadratic Residuosity (QR) + superpolynomial LPN*[2]

This can be extended under the same assumptions to secure 3-party computation of general "layered" (in fact, only locally synchronous[3]*) circuits of depth d*

[2] Superpolynomial hardness of LPN with a small inverse-superpolynomial error rate, but few samples, as assumed in [23].

[3] A circuit is *layered* [31] if all gates and inputs are arranged into layers, such that any wire only connects one layer to the next, but each input may occur multiple times at different layers. A layered circuit is *locally synchronous* [5] if each input occurs exactly once (but at an arbitrary layer). A locally synchronous circuit is *synchronous* [32] if all inputs are in the first layer.

and size s with communication $O(s/\log\log s + d^{1/3} \cdot s^{2(1+\epsilon)/3} \cdot \mathsf{poly}(\lambda))$, for arbitrary small constant ϵ. The latter is sublinear in s whenever $d = o(s^{1-\epsilon}/\mathsf{poly}(\lambda))$, i.e., the circuit is not too "tall and skinny."

If we further assume the existence of a constant-locality PRG with some polynomial stretch and the super-polynomial security of the Decisional Composite Residuosity (DCR) assumption, then the above extends to the 5-party setting, both for loglog-depth boolean circuits and for layered boolean circuits.

More concretely, the required notion of correlated SPIR was achieved in [9], building on [17], from a selection of different assumptions. The required HSS follows for $N = 2$ from DDH from [11], LWE with polynomial-size modulus from [16,25], DCR from [37,40], and from superpolynomial LPN from [23]. It holds for $N = 4$ from DCR from [19] (with some extra work, complexity leveraging, and restrictions; see technical section). Note that combining the works of [17,37] seems to implicitly yield rate-1 batch OT from DCR, and in turn correlated SPIR [9]: if true, the assumptions for sublinear-communication five-party MPC can be simplifed to constant-locality PRG, LPN, and superpolynomial DCR (without the need for DDH, LWE, or QR). Since this claim was never made formally, we do not use it.

A beneficial consequence of our framework is that future developments within these areas can directly be plugged in to yield corresponding new constructions and feasibilities.

1.2 Technical Overview

General Framework. Recall the secure computation framework via homomorphic secret sharing (HSS). Given access to an N-party HSS scheme supporting homomorphic evaluation of the desired circuit C, the parties begin by jointly HSS-sharing their inputs via a small secure computation. Each party can then homomorphically evaluate the circuit C on its respective HSS share without interaction, resulting in a short output share that it exchanges with all other parties. The parties can then each reconstruct the desired output by combining the evaluated shares (for standard HSS, this operation is simply addition). The resulting MPC communication cost scales only with the complexity of HSS share generation plus exchange of (short) output shares, but remains otherwise independent of the complexity of C.

In theory, this approach provides sublinear secure computation protocols for any number of parties N. In practice, however, we simply do not have HSS constructions for rich function classes beyond $N = 2$ with security against collusion of two or more corrupted parties, crucial for providing the corresponding MPC security. This remains a standing open question that has received notable attention, and unfortunately seems to be a challenging task.

A natural question is whether the above framework can somehow be modified to extend beyond the number of parties N supported by the HSS, for example to $N' = N + 1$. The issue with the above approach is that parties cannot afford to secret share their input to any N-subset in which they do not participate, as

all parties within this subset may be corrupt, in which case combining all HSS shares reveals the shared secrets.

Instead, suppose that only the N parties share their inputs amongst each other. In this case, there is no problem with all N shareholding parties being corrupt, as this reveals only their own set of inputs. But, we now have a challenge: how to involve the final party's input into the computation?

In the HSS framework, parties each homomorphically evaluated the public C on shares. Suppose, on the other hand, the HSS supports homomorphic evaluation of the *class* of functions $C_{x_0} := C(x_0, \cdot, \ldots, \cdot)$. Or, more naturally, consider a dual view: Where the N parties collectively generate shares of a secret *function* $C(\cdot, x_1, \ldots, x_N)$ with their inputs hardcoded, which accepts a single input x_0 and outputs $C(x_0, \ldots, x_N)$. That is, using *function* secret sharing (FSS).

Of course, normally in FSS we think of the input on which the function is to be evaluated (in this case, x_0) as a public value, which each shareholder will know. Here, this clearly cannot be the case. Instead, we consider a modified approach, where each of the N FSS shareholders will perform a pairwise *oblivious evaluation* procedure, with the final $(N+1)$st party P_0. That is, the ith shareholder holds the ith function key FSS k_i, which defines a share evaluation function "f_i"$= \mathsf{FSS.Eval}(i, k_i, \cdot)$. As a result of the oblivious evaluation, party P_0 will learn the evaluation $y_i = \mathsf{FSS.Eval}(i, k_i, x_0)$ of this function on its secret input x_0, and neither party will learn anything beyond this; in particular, P_0 does not learn k_i, and P_i does not learn x_0. At the conclusion of this phase, party P_0 learns exactly the set of N output shares, and can reconstruct the final output $C(x_0, \cdots, x_N) = y_1 + \cdots + y_N$ and send to all parties.

The corresponding high-level protocol template is depicted in Fig. 1. Here, $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ represents an ideal N-party functionality for N-FSS share generation (defined formally in Fig. 2 of Sect. 3), where each party provides its input x_i and receives its FSS share k_i. $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ represents an ideal two-party functionality for oblivious FSS share evaluation (defined formally in Fig. 3 of Sect. 3), where P_i and P_0 respectively provide inputs k_i and x_0, and P_0 learns the evaluation $\mathsf{FSS.Eval}(i, k_i, x_0)$.

Consider the (passive) security of the proposed scheme against up to N corruptions. If the corrupted parties are (any subset of) those holding FSS shares, then since the parties execute a secure computation for share generation, their view is restricted to a subset of FSS key shares $(k_i)_{i \in T}$, which hides any honest parties' inputs $(x_i)_{i \in [N] \setminus T}$ by the security of the FSS. (Note if all N shareholding parties are corrupt, then this statement holds vacuously, as no honest parties' inputs were involved.) If the corrupted parties include P_0 together with a (necessarily *strict*) subset of FSS shareholders, then their collective view consists of a strict subset of FSS keys $(k_i)_{i \in T}$ together with evaluated output shares $(y_i)_{i \in [N]}$. However, the security of the FSS combined with the additive reconstruction of output shares implies this reveals nothing beyond the function output.[4]

[4] Note that in fact we do not need FSS with *additive* reconstruction, but rather any form of reconstruction will suffice, as long as the output shares provide this property of revealing nothing beyond the function output. We formalize this property, and prove it holds for additive reconstruction, in the full version of the paper.

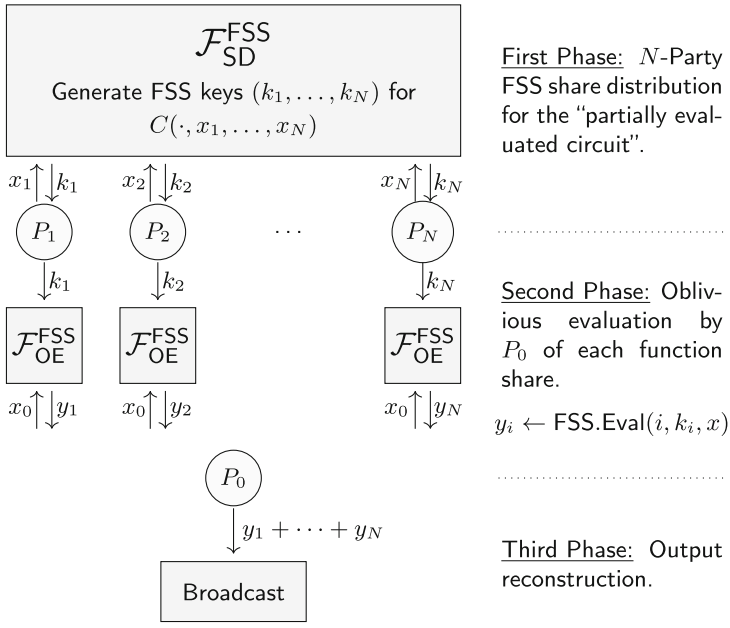

Fig. 1. Template for $(N + 1)$-party sublinear secure computation of C from N-party additive FSS.

Now, in order for this framework to provide low communication, it must be the case that we have an FSS scheme for the relevant partial-evaluation function class $\{f_{\alpha_1,\ldots,\alpha_N} = C(\cdot, \alpha_1, \ldots, \alpha_N)\}$, for which the following two steps can be performed succinctly:

- Secure N-party FSS share generation, and
- Oblivious evaluation by P_0 of each function share.

We next address approaches for how each of these pieces can be achieved.

Oblivious Evaluation for "Loglog-depth" FSS via PIR. Consider first the pairwise oblivious FSS evaluation procedure, where P_0 holds x_0, party P_i holds FSS key k_i, and P_0 should learn $\mathsf{FSS.Eval}(i, k_i, x_0)$.

Since this is reduced to a 2-party functionality, a natural first place to look would be for FSS schemes where $\mathsf{FSS.Eval}(i, k_i, \cdot)$ is within a function class already admitting low-communication *2-party* secure computation. Unfortunately, this is more challenging than it sounds. While sublinear-communication 2PC exists for general layered circuits from a variety assumptions, recall that the sublinearity will be here in the complexity not of C, but of $\mathsf{FSS.Eval}$, almost certainly a more complex computation.

Indeed, the idea of increasing the number of parties by homomorphically evaluating an $\mathsf{HSS.Eval}$ has previously been considered in the related setting of HSS, and hit similar limitations. For example, relatively strong HSS schemes based

on DDH or DCR support homomorphic evaluation (and thus secure computation with very low communication) of NC^1; but, the corresponding operations required to actually *compute* $\mathsf{HSS.Eval}$ itself lies outside of NC^1. In [13], this was addressed by instead securely computing a (low-depth) *randomized encoding* of the evaluation operation, effectively squashing the depth of the computation to be securely performed. This enabled them to achieve low round complexity, but resulted in large communication (scaling with the size of the entire $\mathsf{HSS.Eval}$ circuit). Recently, it was shown by Chillotti et al. [19] that for the specific DCR-based HSS construction of [37,40], $\mathsf{HSS.Eval}$ for homomorphically evaluating a constant-degree computation can be computed within NC^1. However, this only gives low-communication secure computation for constant-degree functions, which will not suffice for overall sublinearity.

Instead, we take a different approach, going beyond black-box use of existing sublinear 2PC results. While the full $\mathsf{FSS.Eval}(i, k_i, x_0)$ computation itself may be complex, suppose it is the case that it can be decomposed into two parts: (1) some form of precomputation, depending only on i and k_i, followed by (2) computation on x_0, which is of low complexity. More concretely, consider the output of part (1) to be a new circuit C_{Eval} whose input is x_0 and output is $\mathsf{FSS.Eval}(i, k_i, x_0)$, and suppose it is the case that C_{Eval} has low $\log\log(s)$ depth (where s is the size of the original circuit C the parties wish to compute in the MPC). Note that while C_{Eval} has low depth, its identity depends on the secret k_i (of P_i), so that black-box secure computation of C_{Eval} does not apply.

On the other hand, opening the box of one such recent secure computation protocol, we identify that an intermediate tool developed actually has stronger implications. The tool is *correlated batch symmetric PIR*, for short correlated SPIR [9], which as discussed above, enables low-communication of several batched instances of (single-server) SPIR whose queries are correlated. In this case, the m "databases" will be defined implicitly by the m output bits of the circuit C_{Eval}. Because C_{Eval} is $\log\log s$ depth as a function of its input x_0 (and circuits are taken to be fan-in 2), each computed output bit depends on at most $\log s$ bits of x_0, and as such can be represented as a size-s database indexed by the corresponding $\log s$ input bits. Oblivious evaluation of C_{Eval} on x_0 can then be achieved by P_0 making m batch queries into these databases, where the collective query bits are all derived from various bits of the single string x_0.

As a brief aside: Extending to larger arithmetic spaces, the role of correlated SPIR here can be replaced by an analogous version of correlated Oblivious Polynomial Evaluation (OPE). Here, a $\log\log s$ depth arithmetic circuit C_{Eval} can be expressed as a secret multivariate polynomial in x_0 of size $\mathsf{poly}(s)$, where each monomial depends on at most $\log s$ elements of the arithmetic vector x_0. Unfortunately, we are not presently aware of tools for achieving low-communication correlated OPE beyond constant-size fields. However, we include this in the technical exposition, in case such techniques are later developed. We note that the final steps in our instantiation (described in the following) do hold over larger arithmetic spaces under certain computational assumptions.

"Loglog-depth" FSS from HSS. Consider an ensemble $\mathcal{C} = \{\mathcal{C}_\lambda\}$ of Boolean circuits of size s and depth $\log\log s$. The remaining goal is to obtain FSS for the corresponding class of partial-evaluation functions $\{C_\lambda(\cdot, x_1, \ldots, x_N)\}$ for which the FSS evaluation C_{Eval} is $(\log\log s)$-depth, as discussed above.

From the structure of C_λ, the evaluation of $C_\lambda(x_0, \ldots, x_N)$ on *all* inputs can be expressed as a $\mathsf{poly}(s)$-size multivariate polynomial in the bits $x_i[j]$ of the x_i, where each monomial is of degree at most $\log s$. When viewed as a function of just x_0, we thus have $\mathsf{poly}(s)$-many monomials in the bits of x_0 whose coefficients p_j are each formed by the product of at most $\log s$ bits from the inputs $x_1, \ldots, x_N$. That is, $\sum_j p_j \prod_{\ell \in S_j} x_0[\ell]$, where each $|S_j| \leq \log s$ is a publicly known set.

If the N parties can somehow produce *additive secret shares* $\{p_j^{(i)}\}_{i\in[N]}$ of each one of these coefficients p_j, then this would constitute the desired FSS evaluation: Indeed, the ith share evaluation $\mathsf{FSS.Eval}(i, k_i, x_0)$ would be computable as $y^{(i)} = \sum_j p_j^{(i)} \prod_{\ell \in S_j} x_0[\ell]$, satisfying $\sum_{i=1}^N y^{(i)} = \sum_j p_j \prod_{\ell \in S_j} x_0[\ell] = C_\lambda(x_0, \ldots, x_N)$. Further, each $\mathsf{FSS.Eval}(i, k_i, \cdot)$ is expressible as a $(\log\log s)$-depth circuit in x_0—as required from the previous discussion.

The question is how to *succinctly* reach a state where the N parties hold these coefficient secret shares. Of course, direct secure computation is not an option, as even the output size is large, $\mathsf{poly}(s)$. However, this is not a general computation. Suppose we have access to an *HSS* scheme supporting homomorphic evaluation of $\log\log s$ depth operations. Such constructions are known to exist from a variety of assumptions (as discussed after Corollary 5). Then, if the parties HSS share their respective inputs $x_1, \ldots, x_N$, they can *locally* evaluate additive shares of the corresponding $(\log s)$-products p_j.

The corresponding $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ operation will thus correspond to the $\mathsf{HSS.Share}$ procedure of the HSS scheme on the parties' collective inputs. If the HSS scheme has a compact sharing procedure, then this will be computable with sufficiently low communication. Note that vanilla usage of some HSS schemes will not provide the required compactness (e.g., including structured ciphertexts of the input bits); however, using standard hybrid encryption tricks this can be facilitated.

"Loglog-depth" FSS from Las Vegas HSS. An additional challenge arises, however, when the underlying HSS scheme we attempt to use provides correctness only up to *inverse-polynomial error*. This is the case, for example, in known 2-party HSS schemes for NC^1 from DDH [11] or from LWE with polynomial-size modulus [16,25]. In these schemes, the inverse-polynomial error rate δ can be chosen as small as desired, but shows up detrimentally as $1/\delta$ in other scheme parameters (runtime for the DDH scheme; modulus size for LWE).

This means with noticeable probability, the shares of at least one of the coefficients p_j from above will be computed incorrectly. Even worse, as typical in these settings, the parties cannot learn or reveal where errors truly occurred, as this information is dependent on the values of the secret inputs. This remains a problem even if the HSS scheme is "Las Vegas," in the sense that for every error at least one of the parties will identity that a potential-error event has occurred (i.e., will evaluate output share as $\perp$). Even then, the flagging party must not

learn whether an error truly took place, and the other party must not learn that a potential error was flagged.

We present a method for modifying the HSS-based FSS sharing procedure from above, to remove the error in the required homomorphic evaluations, while hiding from the necessary parties where these patches took place. We focus on the 2-party case, and further assume the HSS has a succinct protocol (communication linear in the input size, up to an additive $\mathsf{poly}(\lambda)$ term) for distributing the shares of the HSS, where homomorphic evaluation can take place across different sets of shared values. This is the case for known Las Vegas HSS schemes.

This procedure can be viewed as a modification to either the Share or Eval portion of the FSS. By viewing it as part of $\mathsf{FSS.Share}$, we automatically fit into the framework of the previous sections. Namely, this can be viewed as a new $\mathsf{FSS.Share}$ (or $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$) procedure with relatively large computational complexity (comparable to the truth table of the shared function), but which we show *admits a low-communication secure computation procedure*. We describe the sharing procedure directly via the achieving protocol; the corresponding $\mathsf{FSS.Share}$ procedure can be inferred.

First, note that by taking the inverse-polynomial error rate δ to be sufficiently small, we can guarantee with high probability that the total number of potential-error flags $\perp$ obtained by any party is at most the security parameter, λ. The sharing protocol begins by HSS sharing the inputs $(s_0, s_1) \leftarrow \mathsf{HSS.Share}(x_1, x_2)$ as usual. Then, each party homomorphically evaluates all required values corresponding to shares of each of the coefficients p_j. For each party P_i ($i \in \{1, 2\}$), denote these values in an array T_i, which contains at most λ positions in which $T_i[j] = \perp$. For each such position j^*, $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ sets $T_i[j^*] = 0$, and must now "patch" the missing value. Consider this procedure for party P_1 (P_2 will be reversed).

In order to compute the correct output (i.e., coefficient p_j) in this position, the parties run a small-scale secure protocol that HSS shares the *index* position j^* of each $\perp$ symbol of P_1. This enables them to homomorphically re-evaluate shares of the corresponding coefficient term p_{j^*}, in a way that hides the index j^* from P_2 (note that this computation, with index selection, remains within NC^1). In fact, by re-evaluating this computation λ-many times, then with overwhelming probability, at least one is error-free. By running a small-scale secure computation on these shares, we can assume that the parties hold additive shares of the correct value p_{j^*}.

It would seem the remaining step is for P_1 to somehow learn the correct value p_{j^*} offset by P_2's share $T_2[j^*]$, while keeping j^* hidden from P_2. However, the situation is somewhat more sticky. The problem is that in the original HSS evaluation, P_1 learns not only $\perp$, but also a candidate output share. By receiving the *correct* output share $(p_{j^*} - T_2[j^*])$, party P_1 would learn whether or not an error actually occurred, leaking sensitive information. This means that inherently, P_2 must also modify its share in position j^* as part of the correction procedure. But, this must be done in a way that both hides the identity of j^*, and also does *not* affect the secret sharing across the two parties in other positions.

This will be done in two pieces: (1) P_1 will learn $(T_2[j^*] - r)$, for some secret mask r chosen by P_2; and (2) they will both perform some operation on their local T_i array that offsets the value shared in position j^* by exactly $(p_{j^*} - T_2[j^*])$ while preserving the values shared in all other $j' \neq j^*$.

The first of these tasks can be performed by executing a standard single-server polylogarithmic symmetric PIR protocol, where P_1 acts as client with query index j^*, and P_2 acts as server with the *r-shifted* database $T_2'[j] = T_2[j] - r$, for random r of its choice.

The second task will be performed by a low-communication private increment procedure using *distributed point functions* (DPF): namely, FSS for the class of point functions (equivalently, compressed secret shares of a secret unit vector). Actually, since party P_1 knows the identity of j^*, a weaker tool of punctured PRFs suffice; however, we continue with DPF terminology for notational convenience (both are implied by one-way functions). More concretely, the parties will run a small-scale secure computation protocol on inputs $j^*, (T_2[j^*] - r)$ (held by P_1), the additive shares of p_{j^*}, and r (held by P_2), which outputs short DPF key shares k_1, k_2 to the respective parties, with the property that $\mathsf{DPF.Eval}(1, k_1, j) + \mathsf{DPF.Eval}(2, k_2, j) = 0$ for every $j \neq j^*$, and $= (p_{j^*} - T_2[j^*])$ for $j = j^*$. Each party thus modifies its T_i array by offsetting each position j with the jth DPF evaluation, yielding precisely the required effect.

This procedure is performed for every flag position j^*, and for each party P_1, P_2. (Note that the parties should always perform the above steps λ times, sometimes on dummy values, in order to hide the true number of flagged positions.) The final resulting scheme provides standard FSS correctness guarantees, *removing* the inverse-polynomial error, and thus can be plugged into the approach from above. As mentioned, the new resulting $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ functionality is now a complex procedure, with runtime scaling as the entire truth table size of the shared function. But, the above-described protocol provides a means for securely emulating $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ with *low communication*: scaling just as λ-many small-scale secure computations and PIR executions.

2 Preliminaries

2.1 Assumptions

We assume familiarity with the following computational assumptions, and refer to the full version for more details: Quadratic Residuosity (QR) [30], Learning With Errors (LWE) [39], Learning Parity with Noise (LPN) [7], Decisional Diffie-Hellman (DDH), and Decision Composite Residuosity (DCR) [38].

2.2 Function Secret Sharing and Homomorphic Secret Sharing

We follow the function secret sharing definition of [12], for the specific leakage function which reveals the input and output domain sizes $(1^n, 1^m)$ of the secret function.

Definition 6 (Function Secret Sharing (FSS)). *An N-party* Function Secret-Sharing (FSS) *scheme (with additive reconstruction) for a function family $\mathcal{F}$ is a pair of algorithms* $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ *with the following syntax and properties:*

- $\mathsf{Gen}(1^\lambda, \tilde{f})$ *is a probabilistic polynomial-time* key generation algorithm*, which on input 1^λ (a security parameter) and $\tilde{f} \in \{0,1\}^\star$ (the description of some function $f\colon \{0,1\}^n \to \{0,1\}^m \in \mathcal{F}$), outputs an N-tuple of keys $(k_1, \ldots, k_N)$. Each key is assumed to contain 1^n and 1^m.*
- $\mathsf{Eval}(i, k_i, x)$ *is a deterministic polynomial-time* evaluation algorithm*, which on input $i \in [N]$ (the party index), k_i (a key defining $f_i\colon \{0,1\}^n \to \{0,1\}^m$), and $x \in \{0,1\}^n$ (an input for f_i), outputs a value $y_i \in \{0,1\}^m$ (the value of $f_i(x)$, the i^{th} share of $f(x)$).*
- ***Correctness:*** *For all $\lambda \in \mathbb{N}$, all $f \in \mathcal{F}$ (described by $\tilde{f}$), and all $x \in \{0,1\}^n$,*

$$\Pr\left[y_1 + \cdots + y_N = f(x) \;:\; \begin{array}{c}(k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, \tilde{f}) \\ y_i \leftarrow \mathsf{FSS.Eval}(i, k_i, x),\ i = 1 \ldots N\end{array}\right] = 1 \ .$$

- ***Security:*** *For every set of corrupted parties $\mathcal{D} \subsetneq [N]$, there exists a probabilistic polynomial-time algorithm* $\mathsf{Sim}^{\mathsf{FSS}}$ *(a* simulator*), such that for every sequence of functions $f_1, f_2, \cdots \in \mathcal{F}$ (described by $\tilde{f}_1, \tilde{f}_2, \ldots$), the outputs of the following experiments* $\mathsf{Real}^{\mathsf{FSS}}$ *and* $\mathsf{Ideal}^{\mathsf{FSS}}$ *are computationally indistinguishable:*
  - $\mathsf{Real}^{\mathsf{FSS}}(1^\lambda)$: $(k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda)$*; Output* $(k_i)_{i \in \mathcal{D}}$.
  - $\mathsf{Ideal}^{\mathsf{FSS}}(1^\lambda)$: *Output* $\mathsf{Sim}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^n, 1^m)$.

We consider also the dual notion of *Homomorphic Secret Sharing* [11], in which the roles of input and function are reversed, as well as a weaker variant with only Las Vegas inverse-polynomial correctness error.

3 General Template for $(N+1)$-Party Sublinear Secure Computation from N-Party FSS

In this section we present a generic template for building $(N+1)$-party sublinear secure computation from an N-party additive function secret sharing scheme (for a well-chosen function class) with two specific properties. We require of the FSS scheme that there exist low-communication protocols to realise the following tasks:

- *N-Party Share Distribution:* N servers generate FSS shares of some function of their inputs; the ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ is provided in Fig. 2.
- *Two-Party Oblivious Share Evaluation:* A client obliviously evaluates an FSS share held by a server; the ideal functionality $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ is provided in Fig. 3.

Theorem 7 proves that the protocol provided in Fig. 5 is an $(N+1)$-party secure computation scheme in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid model. This template achieves sublinear secure computation provided $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ and $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ can be realised with low enough communication. A high level overview of the protocol is provided in Fig. 1.

3.1 Requirements of the FSS Scheme

We start by isolating in the properties we require of the FSS scheme to fit our template for sublinear secure computation, and show that they are satisfied by any additive FSS scheme. At a high level, we require that given a strict subset of the FSS keys, together with the evaluated output shares of all keys on some known input x, it should be computationally hard to recover any information about the secret shared function f beyond its evaluation $f(x)$. For the formalisation of this property, as well as the proof that additivity suffices, we refer to the full version of the paper.

3.2 The Secure Computation Protocol

We define the ideal functionalities $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ (Fig. 2) for N-party FSS share distribution, and $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ (Fig. 3) for 2-party oblivious evaluation of FSS shares. We then introduce in Fig. 5 the generic template for secure computation from additive FSS in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid model.

Functionality FSS Share Distribution $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$

Parameters: The ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ is parameterised by a number of parties N, a function class $\mathcal{C} = \{f_{\alpha_1,\ldots,\alpha_N}\}_{(\alpha_1,\ldots,\alpha_N)\in\mathbb{F}^{\ell_1}\times\cdots\times\mathbb{F}^{\ell_N}}$, and an additive FSS scheme $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ for $\mathcal{C}$.

$\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ interacts with the N parties $P_1, \ldots, P_N$ in the following manner.

Input: Wait to receive $(\mathtt{input}, i, x_i)$ where $x_i \in \{0,1\}^{\ell_i}$ from each party P_i (for $1 \le i \le N$).

Output: Run $(k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, \tilde{f}_{x_1,\ldots,x_N})$, where $\tilde{f}_{x_1,\ldots,x_N}$ is a description of $f_{x_1,\ldots,x_N}$; Output k_i to each party P_i (for $1 \le i \le N$).

Fig. 2. Ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ for the generation of FSS keys of a distributed function.

Theorem 7 (Template for $(N+1)$-Party Sublinear MPC from N-Party FSS). *Let $N \ge 2$. Let $C\colon \mathbb{F}^n \to \mathbb{F}^m$ be an arithmetic circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs over a finite field $\mathbb{F}$, and let $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ be an (additive) FSS scheme for the following function family of "partial evaluations of C":*

$$\left\{ g_{\alpha_1,\ldots,\alpha_N}\colon \begin{array}{rl} \mathbb{F}^{\ell_0} & \to \mathbb{F}^m \\ x & \mapsto C(x, \alpha_1, \ldots, \alpha_N) \end{array} : (\alpha_1, \ldots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \right\} .$$

Functionality Oblivious Evaluation of FSS Shares $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$

Parameters: *The ideal functionality* $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ *is parameterised by a number of parties* N*, and an additive FSS scheme* $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ *for some function class* $\mathcal{C}$*.*

$\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ *interacts with two parties, Alice ("the client") and Bob ("the server"), in the following manner.*

Input: *Wait to receive* $(\texttt{Client}, x)$ *from Alice and* $(\texttt{Server}, i, k_i)$ *from Bob, and record* (i, k_i, x)*.*

Output: *Run* $y_i \leftarrow \mathsf{FSS.Eval}(i, k_i, x)$*; Output* y_i *to Alice.*

Fig. 3. *Ideal functionality* $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ *for the two-party oblivious evaluation of FSS shares.*

Functionality $\mathcal{F}_{\mathsf{SFE}}(C)$

The functionality is parameterised with a number N *and an arithmetic circuit* C *with* $n = \ell_0 + \ell_1 + \cdots + \ell_N$ *inputs and* m *outputs over a finite field* $\mathbb{F}$*.*

Input: *Wait to receive* $(\texttt{input}, i, x_i)$ *from each party* P_i $(0 \leq i \leq N)$*, where* $x_i \in \mathbb{F}^{\ell_i}$*, and set* $\boldsymbol{x} \leftarrow x_0 \| x_1 \| \ldots \| x_N$*.*

Output: *Compute* $\boldsymbol{y} \leftarrow C(\boldsymbol{x})$*; Output* $\boldsymbol{y}$ *to all parties* $P_0, P_1, \ldots, P_N$*.*

Fig. 4. *Ideal functionality* $\mathcal{F}_{\mathsf{SFE}}(C)$ *for securely evaluating an arithmetic circuit* C *among* $N + 1$ *parties.*

The protocol Π_C *provided in Fig. 5 UC-securely implements the* $(N+1)$*-party functionality* $\mathcal{F}_{\mathsf{SFE}}(C)$ *in the* $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}(C), \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C))$*-hybrid model, against a static passive adversary corrupting at most* N *out of* $(N+1)$ *parties. The protocol uses* $N \cdot m \cdot \log |\mathbb{F}|$ *bits of communication, and additionally makes one call to* $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}(C)$ *and* N *calls to* $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C)$*.*

We refer the reader to the full version of the paper for the proof of Theorem 7.

Protocol Π_C

Parties: $P_0, P_1, \ldots, P_N$

Parameters: The protocol is parameterised with a number of parties $(N+1)$, an arithmetic circuit $C\colon \mathbb{F}^n \to \mathbb{F}^m$ with $n = \ell_0 + \ell_1 + \cdots + \ell_N$, and an additive FSS scheme $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ for the following function family of "partial evaluations of C":

$$\left\{ \begin{array}{rl} g_{\alpha_1,\ldots,\alpha_N}\colon \mathbb{F}^{\ell_0} & \to \mathbb{F}^m \\ x & \mapsto C(x, \alpha_1, \ldots, \alpha_N) \end{array} : (\alpha_1, \ldots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \right\} .$$

$(\mathsf{sid}_1, \ldots, \mathsf{sid}_N)$ are N distinct session ids.

Hybrid Model: The protocol is defined in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid model.

Input: Each party P_i holds input $x_i \in \mathbb{F}^{\ell_i}$.

The Protocol:

1. Each party P_i for $i \neq 0$ sends $(\mathtt{input}, i, x_i)$ to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}(C)$, and waits to receive k_i.
2. For each $i = 1, \ldots, N$:
   (a) Party P_0 sends $(\mathsf{sid}_i, \mathtt{Client}, x_0)$ to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C)$ and P_i sends $(\mathsf{sid}_i, \mathtt{Server}, i, k_i)$ to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C)$
   (b) Party P_0 waits to receive (sid_i, y_i) from $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C)$.
3. Party P_0 sets $\boldsymbol{y} \leftarrow y_1 + \cdots y_N$, and sends $\boldsymbol{y}$ to all parties.
4. Every party outputs $\boldsymbol{y}$.

Fig. 5. (Sublinear) secure computation protocol in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid.

4 Oblivious Evaluation of LogLog-Depth FSS from PIR

In the previous section we provided a generic template for $(N+1)$-party sublinear secure computation from N-party additive function secret sharing for which $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ and $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ can be securely realised with low communication. In this section we introduce the notion of *loglog-depth* for (additive) FSS schemes, and show that this property allows $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ to be securely realised with low communication using *correlated symmetric PIR (*$\mathsf{corrSPIR}$*)*, a primitive introduced in [9] (and which can be instantiated from standard assumptions using the rate-1 batch OT from [17]).

4.1 LogLog-Depth FSS

A depth-d, n-input, m-output arithmetic circuit with gates of fan-in at most two over a finite field $\mathbb{F}$ can be associated with the degree-($\leq 2^d$) n-variate m-output[5] polynomial with coefficients in $\mathbb{F}$ that it computes. In all generality, a degree-2^d n-variate polynomial can have up to $\mathsf{nb}_{n,2^d} = \sum_{k=0}^{2^d} \binom{k+n-1}{n-1}$ different monomials (which can be verified using a stars-and-bars counting argument). In this section we will only be interested in circuits whose representation as a polynomial is the sum of $\mathsf{poly}(\lambda)$ monomials (where λ is the security parameter). A sufficient condition is for it to have $n = \mathsf{poly}(\lambda)$ inputs and depth $d \leq \log\log(n)$; we refer to this property as a circuit being "of loglog-depth". Indeed, because we only consider circuits whose gates have fan-in at most two, if a circuit has depth d then it is 2^d-local (*i.e.* each of its m outputs is a function of only at most 2^d inputs). Therefore each of its outputs is computed by a polynomial with at most $\mathsf{nb}_{2^d,2^d} \leq 2^{2^d+2^d}$ monomials, which is $\mathsf{poly}(\lambda)$ if $d = \log\log n = \log\log\lambda + \mathcal{O}(1)$.

We extend in Definition 8 the above notion of "loglog-depth" circuits to "loglog-depth" FSS schemes.

Definition 8 (LogLog-Depth FSS). *Let $\mathcal{F}$ be a class of functions with n inputs and m outputs over a finite field $\mathbb{F}$. We say that an N-party FSS scheme* $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ *for $\mathcal{F}$ whose evaluation algorithm* $\mathsf{FSS.Eval}$ *is explicitly described as an arithmetic circuit, has* loglog-depth *(alternatively,* FSS *is a loglog-depth function secret sharing scheme) if for every party index $i \in [N]$ and every key $k_i \in \mathsf{Supp}([\mathsf{FSS.Gen}]_i)$ the circuit $\mathsf{FSS.Eval}(i, k_i, \cdot)$ (which has hardcoded i and k_i) has depth* $\log\log(n)$.

Throughout this section we will be using "loglog-depth" circuits and FSS schemes, but it should be noted that all of our results go through if this is replaced everywhere with the more obtuse notion of "circuits (*resp.* FSS evaluation) whose polynomial representation has a polynomial number of coefficients".

When considering "loglog-depth", which in particular are "log-local" circuits, we will be interested in the log-sized subsets of the inputs on which each output depends. We say that an FSS scheme is $(S_1, \ldots, S_m)$-local if the j^{th} output of $\mathsf{FSS.Eval}$, which takes as input a party index i, a key k_i, and an input x, only depends on $(i, k_i, x[S_j])$. In other words, an FSS scheme is $(S_1, \ldots, S_m)$-local if its evaluation algorithm is $(S_1, \ldots, S_m)$-local in its last input. We refer to the full version of this paper for a more complete treatment. We emphasize that a loglog-depth circuit or FSS scheme is always log-local, but that the converse is not necessarily true if $\mathbb{F} \neq \mathbb{F}_2$.

4.2 Oblivious Evaluation of LogLog-Depth FSS from PIR

We first discuss the notion of PIR we need, then show how it can be leveraged to build oblivious evaluation of any loglog-depth FSS scheme.

[5] An m-output (multivariate) polynomial can be seen as a tuple of m (multivariate) polynomials.

4.2.1 Correlated PIR

We refer to [9] for the definition of the ideal functionality for *batch SPIR with correlated "mix and match" queries* ($\mathcal{F}_{\mathsf{corrSPIR}}$), which we extend in the full version of this paper from the boolean to the arithmetic setting as *batch Oblivious (Multivariate) Polynomial Evaluation with correlated "mix and match" queries* ($\mathcal{F}_{\mathsf{corrOPE}}$).

In the boolean world, this corresponds to a batched form of SPIR, querying into k size-N databases, where the queries are not independent. Rather, the queried indices can be reconstructed via a public function that "mixes and matches" individual bits of a single bitstring $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_w)$ of length $w < k \log N$, in a public manner. What this means is that each of the $(n = \log N)$-bit queries to a single database can be obtained by concatenating n of the bits α_i, possibly permuted. In the arithmetic world, this corresponds to batch multivariate OPE, where each database corresponds to a polynomial, and the evaluation inputs are various subvectors of some *joint input vector*, comprised of w field elements. More specifically, the input to a single d-variate polynomial (in the batch to be obliviously evaluated) is a size-d ordered subset of the joint inputs.

We will be interested in how many times a given bit of entropy (*resp.* input) α_i appears within the k queries (*resp.* input)–counted by the occurrence function t_i below–, as well as how many times it appears in specific index position $j' \in [n]$ within the k queries (*resp.* input)–denoted below by $t_{i,j'}$–. To the best of our knowledge, there are no protocols realising corrOPE over superpolynomial-size fields without FHE, and the only protocol realising corrSPIR without FHE requires introducing this notion of "balance between the queried bits".

We refer to [9] for the formalisation of "mix and match" functions, (the ideal functionality for) batch SPIR with correlated "mix and match" queries ($\mathcal{F}_{\mathsf{corrSPIR}}$), and to the full version of this paper for the (ideal functionality for) batch Oblivious (Multivariate) Polynomial Evaluation with correlated "mix and match" queries ($\mathcal{F}_{\mathsf{corrOPE}}$).

4.2.2 Oblivious Evaluation of LogLog-Depth FSS from PIR

Let $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ be a loglog-depth, $(S_1, \ldots, S_m)$-local FSS scheme (Definition 8). Because FSS has loglog-depth, the polynomial representation of FSS.Eval has $m \cdot \mathsf{poly}(n)$ coefficients. Furthermore, each of its local evaluation algorithms $\mathsf{FSS.Eval}_j$ depends only on the inputs indexed by S_j. Therefore obliviously evaluating FSS.Eval can be done by using batch OPE with correlated "mix and match" inputs: the m polynomials in the batch are the $\mathsf{FSS.Eval}_j(i, k_i, \cdot)$, where k_i is known only to the server P_i. This protocol is formalised in Fig. 6.

Note that this notion of corrOPE, as defined in the full version of the paper, requires the polynomials in the batch be represented as a vector of coefficients. For this reason we impose that FSS be loglog-depth, so this vector be polynomial-size.

Protocol Oblivious Evaluation of Partial Function Shares Π_{OE}

Parties: P_0 (the client) and P_i (the server).

Parameters:

- Let N be a number, and let $C = C_1 \| \dots \| C_m$ be a loglog-depth circuit (definition 8) with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs and m outputs over $\mathbb{F}$ such that the following function family $\mathcal{C}$ is $(S_1, \dots, S_m)$-local , where $S_1, \dots, S_m$ is some family of $(\log / \log\log)$-sized subsets of $[n]$:

$$\mathcal{C} = \left\{ \begin{matrix} g_{\alpha_1,\dots,\alpha_N} : \mathbb{F}^{\ell_0} \to \mathbb{F}^m \\ x \mapsto C(x, \alpha_1, \dots, \alpha_N) \end{matrix} : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \right\}.$$

  We assume that each of the S_i is ordered in such a way that the function $\mathsf{MixAndMatch}$ associated with $(S_1, \dots, S_m)$ is $\mathsf{polylog}$-balanced[a] (*c.f.* full version).
- $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ is an $(S_1, \dots, S_m)$-local (additive) FSS scheme for $\mathcal{C}$, whose local evaluation algorithms (*c.f.* full version) are $(\mathsf{FSS.Eval}_j)_{j \in [m]}$.

Hybrid Model: The protocol is defined in the $\mathcal{F}_{\mathsf{corrOPE}}$-hybrid model (the subsets characterising $\mathsf{MixAndMatch}$, and in turn $\mathsf{corrOPE}$, are $(S_1, \dots, S_m)$).

Input: P_0 holds input $x_0 \in \{0,1\}^{\ell_0}$, and P_i holds k_i.

The Protocol:

- **First Round:**
  1. P_0 sends $(\texttt{receiver}, x_0)$ to $\mathcal{F}_{\mathsf{corrOPE}}$
  2. P_i sends $(\texttt{sender}, (\boldsymbol{c}_j)_{j \in [m]})$ to $\mathcal{F}_{\mathsf{corrOPE}}$ where $\boldsymbol{c}_j$ is the vector of coefficients of $\mathsf{FSS.Eval}_j(i, k_i, \cdot)$
     // For the case $\mathbb{F} = \mathbb{F}_2$ (*i.e.* when using $\mathcal{F}_{\mathsf{corrSPIR}}$), the databases can be more simply described as the truth tables of the $\mathsf{FSS.Eval}_j(i, k_i, \cdot)$ for $j \in [m]$, *i.e.* $(\mathsf{FSS.Eval}_j(i, k_i, x'))_{x' \in \{0,1\}^{\lvert S_j \rvert}}$.
  3. **Second Round:**
  4. P_0 waits to receive $(y_{i,1}, \dots, y_{i,m})$ from $\mathcal{F}_{\mathsf{corrOPE}}$
  5. P_0 outputs $(y_{i,1}, \dots, y_{i,m})$

[a] By [9, Lemma 9], such orderings exist and furthermore can be found in expected constant time by random shuffling. Alternatively, since a random ordering of $(S_1, \dots, S_m)$ works with high probability, the protocol could be modified so that P_0 samples a PRG key and sends it to P_1, and both use the resulting pseudorandomness to order $(S_1, \dots, S_m)$. This additional step incurs only a small additive overhead in communication, and the resulting protocol is still sublinear.

Fig. 6. Two-party protocol for obliviously evaluating shares of an additive loglog-depth FSS scheme.

Lemma 9 (Oblivious Share Evaluation for LogLog-Depth FSS Schemes). *Let $N \geq 2$. Let $C\colon \mathbb{F}^n \to \mathbb{F}^m$ be a loglog-depth arithmetic circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs over a finite field $\mathbb{F}$, and let $\mathcal{C}$ be the family of "partial evaluations of C":*

$$\left\{ \begin{array}{rl} g_{\alpha_1,\ldots,\alpha_N}\colon \mathbb{F}^{\ell_0} & \to \mathbb{F}^m \\ x & \mapsto C(x,\alpha_1,\ldots,\alpha_N) \end{array} : (\alpha_1,\ldots,\alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \right\} .$$

If FSS *is an additive loglog-depth, $(S_1,\ldots,S_m)$-local FSS scheme (Definition 8) for $\mathcal{C}$ and* $\mathsf{corrSPIR}$ *is a two-round batch SPIR protocol (characterised by $(S_1,\ldots,S_m)$), then the protocol Π_{OE} provided in Fig. 6 UC-securely implements the two-party functionality $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ against a static, passive adversary in the $\mathcal{F}_{\mathsf{corrOPE}}$-hybrid model.*

The proof of Lemma 9 is given in the full version of the paper.

5 LogLog-Depth FSS from Compact and Additive HSS

In this section we show how to use compact and additive HSS to build a loglog-depth FSS scheme whose share distribution $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ can be realised in low communication. When combined with Sects. 3 to 4, this yields sublinear secure computation from compact and additive HSS. In the full version of this paper, we show how to extend this construction to use the weaker primitive of Las-Vegas HSS. We note that this extension forms a non-trivial technical contribution of our work, which we defer to the full version due to lack of space, and focus this edition of the paper on the simplest version of our template.

5.1 An Overview of the Construction

Let $C\colon \mathbb{F}^n \to \mathbb{F}^m$ be a log log-depth arithmetic circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs over a finite field $\mathbb{F}$, and let $\mathcal{C}$ be the family of "partial evaluations of C":

$$\left\{ \begin{array}{rl} g_{\alpha_1,\ldots,\alpha_N}\colon \mathbb{F}^{\ell_0} & \to \mathbb{F}^m \\ x & \mapsto C(x,\alpha_1,\ldots,\alpha_N) \end{array} : (\alpha_1,\ldots,\alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \right\} .$$

Our goal in this section is to provide a construction of a loglog-depth FSS scheme for $\mathcal{C}$ such that $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ can be realised with low communication, and we do so by using compact and additive single-function HSS for any function in a well-chosen function class (that of $\{\mathsf{coefs}_{c_1,\ldots,c_N} : (c_1,\ldots,c_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$, as defined below).

We provide in Fig. 7 a construction of loglog-depth additive FSS for $\mathcal{C}$ from single-function additive HSS for the following function coefs:

$$\begin{array}{rl} \mathsf{coefs}\colon \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} & \to \mathbb{F}^\star \\ (\alpha_1,\ldots,\alpha_N) & \mapsto (p_0,p_1,\ldots) \end{array}$$

where $(p_0,p_1,\ldots)$ are the coefficients of the polynomial representation of all the $C_j(X,\alpha_1,\ldots,\alpha_N)$, for $j \in [m]$ (which are polynomials in X, whose coefficients

are themselves polynomials in $\alpha_1, \ldots, \alpha_N$). Because C has loglog-depth (Definition 8), there are at most $m \cdot n \cdot (1 + o(1))$ such coefficients. Furthermore, the key generation algorithm of the FSS scheme for $\mathcal{C}$ essentially boils down to a single call to the share generation algorithm of the HSS scheme for coefs. Therefore, we also need to provide an HSS scheme for coefs whose share generation can be distributed using low communication. We use a transformation akin to hybrid encryption in order to ensure this last property: we mask the inputs using pseudorandom generators, and reduce the problem of generating HSS shares of the inputs to that of distributing HSS shares of the keys, which can be done generically using oblivious transfer.

More precisely, for $i \in [N]$ let $G_i \colon \{0,1\}^\lambda \to \mathbb{F}^{\ell_i}$ be a PRG and consider the function family $\{\mathsf{coefs}_{c_1,\ldots,c_N} \colon (c_1, \ldots, c_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$, where:

$$\begin{aligned} \mathsf{coefs}_{c_1,\ldots,c_N} \colon \mathbb{F}^\lambda \times \cdots \times \mathbb{F}^\lambda &\to \mathbb{F}^\star \\ (K_1, \ldots, K_N) &\mapsto (p_0, p_1, \ldots) \end{aligned}$$

where $(p_0, p_1, \ldots)$ are the coefficients of the polynomial representation of all the $C_j(X, c_1 - G_1(K_1), \ldots, c_N - G_N(K_N))$, $j \in [m]$ (which are polynomials in X, whose coefficents are polynomials in the bits of $K_1, \ldots, K_N$).

We provide in the full version of this paper the construction of an HSS scheme for coefs whose share generation can be distributed using low communication (along with a corresponding protocol), assuming the existence of compact and additive single-function HSS for any function in $\{\mathsf{coefs}_{c_1,\ldots,c_N}\}$.

While this assumption relating to the existence of HSS for $\{\mathsf{coefs}_{c_1,\ldots,c_N}\}$ may not seem standard, it is weaker than each of the following assumptions:

1. HSS for NC^1 and polynomial-stretch PRGs in NC^1;
2. Single-function HSS for any $\log\log$-depth circuit and constant-depth PRGs with some fixed polynomial-stretch.

5.2 Defining the LogLog-Depth FSS Scheme

Observation 1 (Parsing Additive Shares). *Let $\boldsymbol{x} \in \{0,1\}^n$ and let $I \subseteq [n]$. If $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$ are additive shares of $\boldsymbol{x}$, then $([\boldsymbol{x}^{(1)}]_I, \ldots, [\boldsymbol{x}^{(m)}]_I)$ are additive shares of $[\boldsymbol{x}]_I$, where $[\cdot]_I$ denotes the subvector induced by the set of coordinates I.*

LogLog-Depth FSS Scheme from Additive HSS

Parameters: Let $N \geq 2$ be a number of parties, and let $C = C_1 \| \dots \| C_m$ be a loglog-depth circuit with $n = \ell_0 + \ell_1 + \dots + \ell_N$ inputs and m outputs over $\mathbb{F}$ such that the following function family is $(S_1, \dots, S_m)$-local, where $S_1, \dots, S_m$ is some family of $(\log n / \log\log n)$-sized subsets of $[n]$:

$$\mathcal{C} = \left\{ \begin{matrix} g_{\alpha_1,\dots,\alpha_N} : \mathbb{F}^{\ell_0} \to \mathbb{F}^m \\ x \mapsto C(x, \alpha_1, \dots, \alpha_N) \end{matrix} : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \right\} .$$

Let $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ be an N-party additive (single-function) HSS scheme for the function:

$$\begin{aligned} \mathsf{coefs} : \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} &\to \mathbb{F}^\star \\ (\alpha_1, \dots, \alpha_N) &\mapsto (p_0, p_1, \dots) \end{aligned}$$

where $(p_0, p_1, \dots)$ are the coefficients of the polynomial representation of all the $C_j(X, \alpha_1, \dots, \alpha_N)$, $j \in [m]$ (which are polynomials in X, whose coefficients are themselves polynomials in $\alpha_1, \dots, \alpha_N$).

// Note that since C has loglog-depth and $\mathcal{C}$ is $(S_1, \dots, S_m)$-local, each of the m polynomials has degree $|S_j|$ and $|S_j|$ variables, and there are therefore at most $\sum_{j=1}^{m} \binom{|S_j|+|S_j|}{|S_j|} = m \cdot n \cdot (1 + o(1))$ coefficients, regardless of $(\alpha_1, \dots, \alpha_N)$.

$\mathsf{FSS.Gen}(1^\lambda, \tilde{g}_{\alpha_1,\dots,\alpha_N})$:

1. Parse $\tilde{g}_{\alpha_1,\dots,\alpha_N}$ to retrieve $(\alpha_1, \dots, \alpha_N)$
2. $(k_1, \dots, k_N) \xleftarrow{\$} \mathsf{HSS.Share}(1^\lambda, i, (\alpha_1, \dots, \alpha_N))$
3. Output $(k_1, \dots, k_N)$

$\mathsf{FSS.Eval}_j(i, k_i, x')$: // $x' \in \mathbb{F}^{|S_j|}$ should be seen as an S_j-subset of some larger $x \in \mathbb{F}^{\ell_0}$ (*i.e.* $x' = x[S_j]$), input of $\mathsf{FSS.Eval}$.

1. $(p_{0,i}, p_{1,i}, \dots) \xleftarrow{\$} \mathsf{HSS.Eval}(i, k_i)$
2. Parse $(p_{0,i}, p_{1,i}, \dots)$ to retrieve shares $(q_{0,i}, q_{1,i}, \dots)$ of the coefficients of $C_j(\cdot, \alpha_1, \dots, \alpha_N)$ (*c.f.* observation 1).
3. $y_{i,j} \leftarrow (x')^{\otimes |S_j|} \cdot (q_{0,i}, q_{1,i}, \dots)^\intercal$
4. Output $y_{i,j}$

$\mathsf{FSS.Eval}(i, k_i, x)$:

1. For $j \in [m]$, set $y_{i,j} \leftarrow \mathsf{FSS.Eval}_j(i, k_i, x[S_j])$
2. Output $(y_{i,j})_{j \in [m]}$

Fig. 7. LogLog-Depth FSS Scheme from "Single-Function" Additive HSS for every LogLog-Depth Circuit.

Lemma 10 (LogLog-Depth FSS Scheme from "Single-Function" Additive HSS). *Let $N \geq 2$ be a number of parties, and let $C = C_1 \| \dots \| C_m$ be a loglog-depth circuit with $n = \ell_0 + \ell_1 + \dots + \ell_N$ inputs and m outputs over $\mathbb{F}$ such that the following function family is $(S_1, \dots, S_m)$-local , where $S_1, \dots, S_m$ is some family of $(\log n / \log \log n)$-sized subsets of $[n]$:*

$$\mathcal{C} = \left\{ \begin{array}{rl} g_{\alpha_1,\dots,\alpha_N} \colon \mathbb{F}^{\ell_0} & \to \mathbb{F}^m \\ x & \mapsto C(x, \alpha_1, \dots, \alpha_N) \end{array} : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \right\} .$$

Let $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *be an N-party (single-function) additive HSS scheme for the function:*

$$\begin{array}{rl} \mathsf{coefs} \colon \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} & \to \mathbb{F}^\star \\ (\alpha_1, \dots, \alpha_N) & \mapsto (p_0, p_1, \dots) \end{array}$$

where $(p_0, p_1, \dots)$ are the coefficients of the polynomial representation of all the $C_j(\cdot, \alpha_1, \dots, \alpha_N)$, $j \in [m]$.

Then the construction of Fig. 7 is an N-party additive loglog-depth and $(S_1, \dots, S_m)$-local FSS scheme for $\mathcal{C}$.

The proof of Lemma 10 is deferred to the full version of the paper.

5.3 Securely Realising $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ in Low Communication

The FSS scheme $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ of Fig. 7 is parameterised by an additive single-function HSS scheme for the function coefs. We provide in the full version of the paper such an HSS scheme with the additional property that it yields FSS for which $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ can be securely realised in low communication.

As explained in the overview of Sect. 5.1, we use a standard hybrid encryption trick in order to build HSS for coefs from HSS for $\{\mathsf{coefs}_{c_1,\dots,c_N}\}$. This allows us to securely distribute the shares of HSS for coefs (and hence the keys of the FSS scheme of Fig. 7) by using generic secure computation to distribute the shares of HSS for $\{\mathsf{coefs}_{c_1,\dots,c_N}\}$ (which can be done in complexity $\mathsf{poly}(\lambda + N)$). We refer to the full version of the paper for the details.

Lemma 11 ($\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ for the LogLog-Depth FSS scheme of fig. 7 can be realised with low communication). *Let $N \geq 2$ be a number of parties, and let $C = C_1 \| \dots \| C_m$ be a loglog-depth circuit with $n = \ell_0 + \ell_1 + \dots + \ell_N$ inputs and m outputs over $\mathbb{F}$ such that the following function family is $(S_1, \dots, S_m)$-local, where $S_1, \dots, S_m$ is some family of $(\log / \log \log)$-sized subsets of $[n]$:*

$$\mathcal{C} = \left\{ \begin{array}{rl} g_{\alpha_1,\dots,\alpha_N} \colon \mathbb{F}^{\ell_0} & \to \mathbb{F}^m \\ x & \mapsto C(x, \alpha_1, \dots, \alpha_N) \end{array} : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \right\} .$$

For $i \in [N]$, let $G_i \colon \{0,1\}^\lambda \to \mathbb{F}^{\ell_i}$ be a constant-depth PRG.

Let $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *be an* N*-party compact and additive HSS scheme for any function in* $\{\mathsf{coefs}_{c_1,\ldots,c_N} : (c_1,\ldots,c_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$*, where:*

$$\mathsf{coefs}_{c_1,\ldots,c_N} : \mathbb{F}^\lambda \times \cdots \times \mathbb{F}^\lambda \to \mathbb{F}^\star \qquad (K_1,\ldots,K_N) \mapsto (p_0, p_1, \ldots)$$

where $(p_0, p_1, \ldots)$ *are the coefficients of the polynomial representation of all the* $C_j(X, c_1 - G_1(K_1), \ldots, c_N - G_N(K_N))$*,* $j \in [m]$ *(which are polynomials in* X*).*

Then the protocol Π_{SD} *provided in the full version of the paper UC-securely implements the* N*-party functionality* $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ *in the* $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$*-hybrid model against a static, passive adversary. Furthermore, assuming oblivious transfer, there exists a protocol (in the real world) using* $(N \cdot \lambda)^{\mathcal{O}(1)} + N(N-1) \cdot n \cdot \log|\mathbb{F}|$ *bits of communication which UC-securely implements the* N*-party functionality* $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ *against a static, passive adversary.*

6 Instantiations

In Sect. 6.1, we combine the results of Sects. 3 to 5 and achieve sublinear secure computation from generic assumptions (HSS and forms of PIR/OLE). In Sect. 6.2, we build four-party compact and additive HSS for loglog-depth correlations from standard assumptions (DCR and constant-locality PRGs). In Sect. 6.3, we show how to combine all the above (as well as existing constructions of 2-party HSS) in order to build sublinear secure 3- and 5-party computation from standard assumptions not previously known to imply it (in particular, they are not known to imply FHE).

6.1 Sublinear-Communication Secure Multiparty Computation from PIR and Additive HSS

Section 4 established that $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ for local FSS schemes can be based on batch OPE (with correlated inputs) and Sect. 5 builds local FSS schemes (such that $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ can be realised with low communication) from additive HSS (with or without errors). Plugging these two constructions into the template of Sect. 3 yields sublinear secure multiparty computation from batch OPE and additive HSS.

Theorem 12 (Sublinear-Communication Secure (N+1)-Party Computation of Shallow Circuits). *Let* $N \geq 2$ *be a number of parties, and let* $C : \mathbb{F}^n \to \mathbb{F}^m$ *be a depth-*d *(*$d \leq \log\log n - \log\log\log n$*) arithmetic circuit with* $n = \ell_0 + \ell_1 + \cdots + \ell_N$ *inputs over* $\mathbb{F}$*. Assuming the existence of:*

- *A family of PRGs* $G_i : \{0,1\}^\lambda \to \mathbb{F}^{\ell_i}$ *for* $i \in [N]$*,*
- *An* N*-party compact and additive single-function HSS scheme for any function in the class* $\{\mathsf{coefs}_{\alpha_1,\ldots,\alpha_N} : (\alpha_1,\ldots,\alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$*, where* $\mathsf{coefs}_{x_1,\ldots,x_N}$ *is the function which, on input* $(K_1,\ldots,K_N) \in (\{0,1\}^\lambda)^N$*, computes the (polynomially many) coefficients of the representation of* $C_j(\cdot, \alpha_1 - G_1(K_1), \ldots, \alpha_N - G_N(K_N))$ *as* ℓ_0*-variate polynomials for* $j = 1$ *to* m*,*
- *A protocol for UC-securely realising* $\mathcal{F}_{\mathsf{corrOPE}}$ *using communication* $\mathsf{Comm}_{\mathsf{corrOPE}}(k, N_{\mathsf{var}}, \deg, w)$*, where* k *is the number of OPEs in the batch,* N_{var} *is the number of variables of each polynomial,* deg *is the degree of each polynomial, and* w *is the size of the joint input vector,*

There exists a protocol using $(N+\lambda)^{\mathcal{O}(1)} + N \cdot [(N-1) \cdot n + m] \cdot \log|\mathbb{F}| + N \cdot \mathsf{Comm}_{\mathsf{corrOPE}}(m, 2^d, 2^d, n)$ *bits of communication to securely compute* C *amongst* $(N+1)$ *parties (that is, to UC-securely realise* $\mathcal{F}_{\mathsf{SFE}}(C)$*) in the presence of a semi-honest adversary statically corrupting any number of parties.*

Proof. The proof of Theorem 12 is obtained by combining the results of Sects. 3 to 5. Our starting point is the generic template of Theorem 7 in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid model, which uses $N \cdot m \cdot \log|\mathbb{F}|$ bits of communication and makes a single call to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$, and N to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$. We use the FSS scheme of lemmas 10 to 11, for which, by Lemma 9, each call to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ can be implemented using communication $\mathsf{Comm}_{\mathsf{corrOPE}}(m, 2^d, 2^d, n)$ and the single call to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ can be implemented using communication $(N \cdot \lambda)^{\mathcal{O}(1)} + N(N-1) \cdot n \cdot \log|\mathbb{F}|$. □

6.2 Four-Party Additive HSS from DCR

In this section, we build a 4-party compact homomorphic secret sharing scheme for the class of loglog-depth circuits. Our starting point is the (non compact) 4-party HSS for constant degree polynomials recently described in [19]. At a high level, the scheme works by *nesting* a 2-party HSS scheme inside another 2-party HSS scheme. Concretely, let $\mathsf{HSS}_{\mathsf{in}}$ and $\mathsf{HSS}_{\mathsf{out}}$ be two 2-party HSS schemes. Then, the following is a 4-party HSS:

- $\mathsf{HSS.Share}(x)$: run $(x^{(0)}, x^{(1)}) \leftarrow \mathsf{HSS.Share}_{\mathsf{in}}(x)$. For $b = 0, 1$, run $(x^{(b,0)}, x^{(b,1)}) \leftarrow \mathsf{HSS.Share}_{\mathsf{out}}(x^{(b)})$. Output $(x^{(0,0)}, x^{(0,1)}, x^{(1,0)}, x^{(1,1)})$.
- $\mathsf{HSS.Eval}(i, f, x^{(i)})$: parse i as $(b, c) \in \{0,1\}^2$. Define $G_{\mathsf{in}}(f) : x^{(b,c)} \rightarrow \mathsf{HSS}_{\mathsf{in}}.\mathsf{Eval}(b, f, x^{(b,c)})$ and run $y^{(i)} \leftarrow \mathsf{HSS}_{\mathsf{out}}.\mathsf{Eval}(c, G_{\mathsf{in}}(f), x^{(b,c)})$.

Therefore, to get 4-party HSS for a function class $\mathcal{F}$, we need (1) a 2-party $\mathsf{HSS}_{\mathsf{in}}$ for $\mathcal{F}$, and (2) a 2-party $\mathsf{HSS.out}$ for the class $\mathcal{F}' = G_{\mathsf{in}}(\mathcal{F})$. We refer to the full version of the paper for how to obtain these two building blocks, and now state the resulting theorem in Theorem 13.

Theorem 13 (Four-Party Additive HSS for Constant-Depth Circuits from DCR). *Assuming the superpolynomial hardness of* DCR *and the existence of PRGs with constant locality, there exists a four-party HSS scheme for the class of loglog-depth circuits with* n_{in} *inputs; the HSS scheme has share size* $n_{in}(1 + o(1))$*. Furthermore, there exists a protocol with communication complexity* $n_{in} \cdot (4 + o(1))$ *(for large enough* n_{in}*) for securely realising the four-party functionality* $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ *for the generation of HSS shares of the concatenation of the parties inputs.*

6.3 Sublinear-Communication Secure Multiparty Computation from New Assumptions

Combining Sect. 6.1 with instantiations of corrSPIR and additive HSS from the literature (and Sect. 6.2) yields sublinear-communication secure 3- and 5-party computation of shallow boolean circuits from a variety of assumptions. Layered boolean circuits are boolean circuits whose gates can be arranged into layers such that any wire connects adjacent layers. It is well-known from previous works [11,22,23] that sublinear protocols for low-depth circuits translate to sublinear protocols for general layered circuits: the parties simply cut the layered circuit into low-depth "chunks", and securely evaluate it chunk-by-chunk. For each chunk, a sublinear secure protocol is invoked to compute the low-depth function which maps shares of the values on the first layer to shares of the values on the first layer of the next chunk.

Theorem 14 (Secure $(N+1)$-Party Computation with Sublinear Communication from New Assumptions).

- **3-PC of Shallow Circuits:** *Let $C\colon \{0,1\}^n \to \{0,1\}^m$ be a size-s, depth-d ($d \leq \log\log s - \log\log\log s$) boolean circuit. Let $\epsilon \in (0,1)$. Assuming the Learning Parity with Noise (LPN) assumption with dimension $\mathsf{dim} = \mathsf{poly}(\lambda)$, number of samples $\mathsf{num} = (n+m)^{1/3} \cdot \lambda^{\mathcal{O}(1)}$, and noise rate $\rho = \mathsf{num}^{\epsilon-1}$ (for some constant $0 < \epsilon < 1$) together with any of the following additional computational assumptions:*
  - *Decisional Diffie-Hellman*
  - *Learning with Errors with polynomial-size modulus*
  - *Quadratic Residuosity and Superpolynomial $\mathbb{F}_2$-LPN (*i.e. *assuming the security against time-$\lambda^{2\log\lambda}$ adversaries of $\mathbb{F}_2$-LPN with dimension $\lambda^{\log\lambda}$, $2\lambda^{\log\lambda}$ samples, and rate $\lambda/(2\lambda^{\log\lambda})$).*

  There exists a 3-party protocol with communication complexity $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n+m+2^{d+2^d} \cdot \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3}))$ to securely compute C (that is, to UC-securely realise $\mathcal{F}_{\mathsf{SFE}}(C)$) in the presence of a semi-honest adversary statically corrupting any number of parties. In particular, if $d \leq (\log\log s)/4$ the communication complexity is $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n+m+\sqrt{s} \cdot \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3}))$ (for some arbitrarily small constant $0 < \delta < 1/2$), which is sublinear in the circuit-size, as detailed in Remark 15.
- **3-PC of Layered Boolean Circuits:** *Let $C\colon \{0,1\}^n \to \{0,1\}^m$ be a size-s, depth-d layered boolean circuit. Let $\epsilon \in (0,1)$. Assuming the Learning Parity with Noise (LPN) assumption with dimension $\mathsf{dim} = \mathsf{poly}(\lambda)$, number of samples $\mathsf{num} = ((s/d)^2/s^{\epsilon})^{1/3} \cdot \mathsf{poly}(\lambda)$, and noise rate $\rho = \mathsf{num}^{-1/2}$ together with any of the following additional computational assumptions:*
  - *Decisional Diffie-Hellman*
  - *Learning with Errors with polynomial-size modulus*
  - *Quadratic Residuosity and Superpolynomial $\mathbb{F}_2$-LPN (*i.e. *assuming the security against time-$\lambda^{2\log\lambda}$ adversaries of $\mathbb{F}_2$-LPN with dimension $\lambda^{\log\lambda}$, $2\lambda^{\log\lambda}$ samples, and rate $\lambda/(2\lambda^{\log\lambda})$).*

  *There exists a 3-party protocol with communication complexity $\mathcal{O}(n+m+d^{1/3} \cdot s^{2(1+\epsilon)/3} \cdot \mathsf{poly}(\lambda) + s/(\log\log s))$ to securely compute C (that is, to UC-securely realise $\mathcal{F}_{\mathsf{SFE}}(C)$) in the presence of a semi-honest adversary statically corrupting any number of parties. In particular, if $d = o(s^{1-\epsilon}/\mathsf{poly}(\lambda))$ (*i.e. *the circuit is not too "tall and skinny") the communication complexity is $\mathcal{O}(n+m+\frac{s}{\log\log s})$, which is sublinear in the circuit-size.*
- **5-PC of Shallow Circuits:** *Let $\epsilon \in (0,1)$. Assuming the existence of a constant-locality PRG with polynomial stretch, there exists a constant $c \geq 3$ such that for any boolean circuit $C\colon \{0,1\}^n \to \{0,1\}^m$ of size s and depth d ($d \leq (\log\log s - \log\log\log s)/2^c$), assuming the superpolynomial Decision Composite Residuosity (DCR) assumption, the Learning Parity with Noise (LPN) assumption with dimension $\mathsf{dim} = \mathsf{poly}(\lambda)$, number of samples $\mathsf{num} = (n+m)^{1/3} \cdot \lambda^{\mathcal{O}(1)}$, and noise rate $\rho = \mathsf{num}^{\epsilon-1}$ (for some constant $0 < \epsilon < 1$), as well as any of the following computational assumptions:*
  - *Decisional Diffie-Hellman (DDH)*
  - *Learning with Errors with polynomial-size modulus (poly-modulus LWE)*
  - *Quadratic Residuosity (QR) and Superpolynomial $\mathbb{F}_2$-LPN (*i.e. *assuming the security against time-$\lambda^{2\log\lambda}$ adversaries of $\mathbb{F}_2$-LPN with dimension $\lambda^{\log\lambda}$, $2\lambda^{\log\lambda}$ samples, and rate $\lambda/(2\lambda^{\log\lambda})$).*

There exists a 5-party protocol with communication complexity $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + 2^{d/2^c+2^{d/2^c}} \cdot \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3}))$ *to securely compute* C *(that is, to UC-securely realise* $\mathcal{F}_{\mathsf{SFE}}(C)$*) in the presence of a semi-honest adversary statically corrupting any number of parties. In particular, if* $d \leq (\log\log s)/2^{c+2}$ *the communication complexity is* $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + \sqrt{s} \cdot \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(n) \cdot ((n + m)^{2/3} + (n + m)^{(1+2\epsilon)/3}))$ *(for some arbitrarily small constant* $0 < \epsilon < 1/2$*), which is sublinear in the circuit-size, as detailed in Remark 15.*

- **5-PC of Layered Boolean Circuits:** *Let* $\epsilon \in (0, 1)$*. Assuming the existence of a constant-locality PRG with polynomial stretch, there exists a constant* $c \geq 3$ *such that for any layered boolean circuit* $C\colon \{0,1\}^n \to \{0,1\}^m$ *of size* s *and depth* d*, assuming the superpolynomial Decision Composite Residuosity (DCR) assumption, assuming the Learning Parity with Noise (LPN) assumption with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = ((s2^c/d)^2/s^\epsilon)^{1/3} \cdot \mathsf{poly}(\lambda)$*, and noise rate* $\rho = \mathsf{num}^{-1/2}$ *together with any of the following additional computational assumptions:*
  - *Decisional Diffie-Hellman*
  - *Learning with Errors with polynomial-size modulus*
  - *Quadratic Residuosity and Superpolynomial* $\mathbb{F}_2$-LPN *(*i.e. *assuming the security against time-*$\lambda^{2\log\lambda}$ *adversaries of* $\mathbb{F}_2$-LPN *with dimension* $\lambda^{\log\lambda}$*,* $2\lambda^{\log\lambda}$ *samples, and rate* $\lambda/(2\lambda^{\log\lambda})$*).*

  There exists a 5-party protocol with communication complexity $\mathcal{O}(n + m + d^{1/3} \cdot s^{2(1+\epsilon)/3} \cdot \mathsf{poly}(\lambda) + s/(\log\log s))$ *to securely compute* C *(that is, to UC-securely realise* $\mathcal{F}_{\mathsf{SFE}}(C)$*) in the presence of a semi-honest adversary statically corrupting any number of parties. In particular, if* $d = o(s^{1-\epsilon}/\mathsf{poly}(\lambda))$ *(*i.e. *the circuit is not too "tall and skinny") the communication complexity is* $\mathcal{O}(n + m + \frac{s}{\log\log s})$*, which is sublinear in the circuit-size.*

Note that combining the works of [17,37] seems to implicitly yield rate-1 batch OT from DCR, and in turn correlated SPIR [9]: if true, the assumptions for sublinear-communication five-party MPC can be simplifed to constant-locality PRG, LPN, and superpolynomial DCR (without the need for DDH, LWE, or QR). Since this claim was never made formally, we do not use it.

The proof of Theorem 14 is deferred to the full version of the paper. We conclude by remarking that while this may not be immediately apparent due to the complicated expressions, the above communication complexities do indeed qualify as "sublinear in the circuit-size".

Remark 15 (The Expressions of Theorem 14 are Sublinear in the Circuit Size). Recall that a protocol for securely computing a size-s circuit with n inputs and m outputs is *sublinear in the circuit-size* if its communication complexity is of the form $\lambda^{\mathcal{O}(1)} + \mathsf{poly}(n + m) + o(s)$, where poly is some fixed polynomial. The communication of our protocols for loglog-depth circuits, both in the 3- and the 5-party case, are sublinear in the circuit-size. For 3PC and 5PC of loglog-depth circuits, the expression is the following:

$$\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + \sqrt{s} \cdot \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(n) \cdot ((n + m)^{2/3} + (n + m)^{(1+2\epsilon)/3})).$$

where $\epsilon \in (0, 1)$ is some constant tied to the strength of the LPN assumption. Because we view s as an arbitrarily large polynomial in the security parameter (in other words we are interested in an asymptotic notion of sublinearity), there exists an arbitrarily

small constant $\delta \in (0, \frac{1}{2})$ such that $\mathsf{poly}(\lambda) \leq s^{\delta}$. Therefore the complexity can be simplified as:

$$\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + s^{\frac{1}{2}+\delta} \cdot \mathsf{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3})).$$

Whenever $s^{\delta} \geq \mathsf{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3})$, the above expression is $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + s^{1+2\delta})$. Whenever $s^{\delta} < \mathsf{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3})$, the entire expression is already some fixed polynomial in $n + m$. Therefore, our final complexity is of the form $\lambda^{\mathcal{O}(1)} + \mathsf{poly}_{\delta}(n+m) + s^{\frac{1}{2}+2\delta}$.

Acknowledgments. We thank the anonymous reviewers of Eurocrypt 2023 for their helpful comments and careful proofreading, which helped to improve the paper. Elette Boyle and Pierre Meyer were supported by AFOSR Award FA9550-21-1-0046, a Google Research Award, and ERC Project HSS (852952). Geoffroy Couteau was supported by the French Agence Nationale de la Recherche (ANR), under grant ANR-20-CE39-0001 (project SCENE), and by the France 2030 ANR Project ANR22-PECY-003 SecureCompute.

References

1. Abram, D., Damgård, I., Orlandi, C., Scholl, P.: An algebraic framework for silent preprocessing with trustless setup and active security. Cryptology ePrint Archive (2022)
2. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_29
3. Barkol, O., Ishai, Y.: Secure computation of constant-depth circuits with applications to database search problems. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 395–411. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_24
4. Beaver, D., Feigenbaum, J., Kilian, J., Rogaway, P.: Security with low communication overhead. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 62–76. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-38424-3_5
5. Belaga, E.G.: Locally synchronous complexity in the light of the trans-box method. In: Fontet, M., Mehlhorn, K. (eds.) STACS 1984. LNCS, vol. 166, pp. 129–139. Springer, Heidelberg (1984). https://doi.org/10.1007/3-540-12920-0_12
6. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC, pp. 1–10. ACM Press (1988). https://doi.org/10.1145/62212.62213
7. Blum, A., Furst, M., Kearns, M., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 278–291. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_24
8. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8270, pp. 280–300. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42045-0_15

9. Boyle, E., Couteau, G., Meyer, P.: Sublinear secure computation from new assumptions. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022, Part II. LNCS, vol. 13748, pp. 121–150. Springer, Heidelberg (Nov 2022). https://doi.org/10.1007/978-3-031-22365-5_5
10. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_12
11. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_19
12. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: improvements and extensions. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 1292–1303. ACM Press (2016). https://doi.org/10.1145/2976749.2978429
13. Boyle, E., Gilboa, N., Ishai, Y.: Group-based secure computation: optimizing rounds, communication, and computation. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 163–193. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_6
14. Boyle, E., Gilboa, N., Ishai, Y., Lin, H., Tessaro, S.: Foundations of homomorphic secret sharing. In: Karlin, A.R. (ed.) ITCS 2018, vol. 94, pp. 1–21. LIPIcs (2018). https://doi.org/10.4230/LIPIcs.ITCS.2018.21
15. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 501–519. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54631-0_29
16. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 3–33. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_1
17. Brakerski, Z., Branco, P., Döttling, N., Pu, S.: Batch-OT with optimal rate. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 157–186. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-07085-3_6
18. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th ACM STOC, pp. 11–19. ACM Press (1988). https://doi.org/10.1145/62212.62214
19. Chillotti, I., Orsini, E., Scholl, P., Smart, N.P., Leeuwen, B.V.: Scooby: improved multi-party homomorphic secret sharing based on FHE. SCN 2022 (2022). https://eprint.iacr.org/2022/862
20. Chor, B., Gilboa, N.: Computationally private information retrieval (extended abstract). In: 29th ACM STOC, pp. 304–313. ACM Press (1997). https://doi.org/10.1145/258533.258609
21. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: 36th FOCS, pp. 41–50. IEEE Computer Society Press (1995). https://doi.org/10.1109/SFCS.1995.492461
22. Couteau, G.: A note on the communication complexity of multiparty computation in the correlated randomness model. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 473–503. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_17
23. Couteau, G., Meyer, P.: Breaking the circuit size barrier for secure computation under Quasi-polynomial LPN. In: Canteaut, A., Standaert, F.-X. (eds.) EURO-

CRYPT 2021. LNCS, vol. 12697, pp. 842–870. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_29
24. Damgård, I., Faust, S., Hazay, C.: Secure two-party computation with low communication. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 54–74. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_4
25. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 93–122. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_4
26. Fazio, N., Gennaro, R., Jafarikhah, T., Skeith, W.E.: Homomorphic secret sharing from Paillier encryption. In: Okamoto, T., Yu, Y., Au, M.H., Li, Y. (eds.) ProvSec 2017. LNCS, vol. 10592, pp. 381–399. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68637-0_23
27. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC, pp. 169–178. ACM Press (2009). https://doi.org/10.1145/1536414.1536440
28. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_35
29. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC, pp. 218–229. ACM Press (1987). https://doi.org/10.1145/28395.28420
30. Goldwasser, S., Micali, S.: Probabilistic encryption and how to play mental poker keeping secret all partial information. In: 14th ACM STOC, pp. 365–377. ACM Press (1982). https://doi.org/10.1145/800070.802212
31. Gál, A., Jang, J.T.: The size and depth of layered boolean circuits. Inf. Process. Lett. **111**(5), 213–217 (2011). https://doi.org/10.1016/j.ipl.2010.11.023
32. Harper, L.H.: An log lower bound on synchronous combinational complexity (1977)
33. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 669–684. ACM Press (2013). https://doi.org/10.1145/2508859.2516668
34. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC, pp. 723–732. ACM Press (1992). https://doi.org/10.1145/129712.129782
35. Kushilevitz, E., Ostrovsky, R.: Replication is NOT needed: SINGLE database, computationally-private information retrieval. In: 38th FOCS, pp. 364–373. IEEE Computer Society Press (1997). https://doi.org/10.1109/SFCS.1997.646125
36. Naor, M., Nissim, K.: Communication preserving protocols for secure function evaluation. In: 33rd ACM STOC, pp. 590–599. ACM Press (2001). https://doi.org/10.1145/380752.380855
37. Orlandi, C., Scholl, P., Yakoubov, S.: The rise of Paillier: homomorphic secret sharing and public-key silent OT. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12696, pp. 678–708. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_24
38. Paillier, P.: Public-key cryptosystems based on composite degree Residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16
39. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC, pp. 84–93. ACM Press (2005). https://doi.org/10.1145/1060590.1060603

40. Roy, L., Singh, J.: Large message homomorphic secret sharing from DCR and applications. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12827, pp. 687–717. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84252-9_23
41. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS, pp. 162–167. IEEE Computer Society Press (1986). https://doi.org/10.1109/SFCS.1986.25