Sublinear-Communication Secure Multiparty Computation Does Not Require FHE

Elette Boyle, Geoffroy Couteau, Pierre Meyer

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. pierre.meyer@irif.fr
[2] NTT Research, Sunnyvale, USA. eboyle@alum.mit.edu
[3] Université Paris Cité, CNRS, IRIF, Paris, FRANCE. couteau@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

Table of Contents

Sublinear-Communication Secure Multiparty Computation does not require FHE 1
Elette Boyle, Geoffroy Couteau, and Pierre Meyer

1 Introduction **3**
1.1 Our Results 3
1.2 Technical Overview 6

2 Preliminaries **10**
2.1 Assumptions 10
2.1.1 Quadratic Residuosity Assumption (QR) 10
2.1.2 Learning With Errors (LWE) 11
2.1.3 Learning Parity with Noise (LPN) 11
2.1.4 Decisional Diffie-Hellman (DDH) 11
2.1.5 Decision Composite Residuosity Assumption 11
2.2 Function Secret Sharing and Homomorphic Secret Sharing 12
2.3 Universal Composability 13
2.4 Notations 13

3 General Template for $(N+1)$-Party Sublinear Secure Computation from N-Party FSS **14**
3.1 Requirements of the FSS Scheme 14
3.2 The Secure Computation Protocol 15

4 Oblivious Evaluation of LogLog-Depth FSS from PIR **18**
4.1 LogLog-Depth FSS 18
4.2 Oblivious Evaluation of LogLog-Depth FSS from PIR 19
4.2.1 Correlated PIR. 19
4.2.2 Oblivious Evaluation of LogLog-Depth FSS from PIR. 21

5 LogLog-Depth FSS from Compact and Additive HSS **22**
5.1 From Compact and Additive HSS 22
5.1.1 An Overview of the Construction 22
5.2 Defining the LogLog-Depth FSS Scheme. 23
5.2.1 Securely Realising $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ in Low Communication. 25
5.2.2 Building HSS for coefs. 25
5.2.3 Low-Communication Protocol for $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ 26
5.3 From Compact and Additive HSS with Errors 28
5.3.1 Additive FSS Scheme from Las Vegas Additive HSS. 29
5.3.2 Securely Realising Gen in Low Communication. 31

6 Instantiations **33**
6.1 Sublinear-Communication Secure Multiparty Computation from PIR and Additive HSS 33
6.2 Four-Party Additive HSS from DCR 34
6.2.1 4-party HSS from DCR. 34
6.2.2 Handling loglog-depth circuits. 35
6.2.3 Compactness and succinct share distribution. 35
6.3 Sublinear-Communication Secure Multiparty Computation from New Assumptions . . 36

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 [Yao86, GMW87, BGW88, CCD88], 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 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 [BFKR91, NN01], or were limited to very simple functions (such as point functions [CGKS95, KO97, CG97] or constant-depth circuits [BI05]). This changed with the breakthrough result of Gentry [Gen09] 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 [DFH12, AJL+12].

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) [BGI16a]. 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 [BGI16a], Decision Composite Residuosity (DCR) [FGJI17, OSY21, RS21], and further algebraic structures, including a class of assumptions based on class groups of imaginary quadratic fields [ADOS22]. It was extended to a flavour of the Learning Parity with Noise (LPN) assumption (via HSS for log log-depth circuits) by [CM21]. Orthogonally to these approaches, which rely on computational assumptions, [Cou19] built sublinear-communciation secure computation under an assumption of correlated randomness.

Very recently, a work of [BCM22] 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" [GI14, BGI15]), where to this date an exponential gap remains between the best constructions in the 2-party versus 3-party setting [BGI15]. 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) [DHRW16], or from subexponentially secure *indistinguishability obfuscation* [BGI15]. Additionally, [BGI+18] turns this FSS from spooky encryption into additive HSS. 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) [BGI15], 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. [BGI+18] for discussion.[4]) 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 $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ 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 [BCM22] 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 [BBDP22]) 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 [BGI16a], or Learning With Errors with only a polynomial-size modulus [DHRW16, BKS19]. This error must be removed in our construction while incurring minimal additional interaction. We demonstrate how to do so, using (standard) Private Information Retrieval [CGKS95] and punctured pseudorandom functions [BW13, KPTZ13, BGI14]. Note that the

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

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 [GMW87] with zero-knowledge arguments, instantiating the ZKA with (interactive) succinct arguments [NN01]. Using Kilian's PCP-based 4-move argument [Kil92], 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. [BGI16a, CM21, BCM22]).

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*[5]

This can be extended under the same assumptions to secure 3-party computation of general "layered" (in fact, only locally synchronous[6]*) circuits of depth d 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.

[5] Superpolynomial hardness of LPN with a small inverse-superpolynomial error rate, but few samples, as assumed in [CM21].

[6] A circuit is *layered* [GJ11] 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* [Bel84] if each input occurs exactly once (but at an arbitrary layer). A locally synchronous circuit is *synchronous* [Har77] if all inputs are in the first layer.

More concretely, the required notion of correlated SPIR was achieved in [BCM22], building on [BBDP22], from a selection of different assumptions. The required HSS follows for $N = 2$ from DDH from [BGI16a], LWE with polynomial-size modulus from [DHRW16,BKS19], DCR from [OSY21, RS21], and from superpolynomial LPN from [CM21]. It holds for $N = 4$ from DCR from [COS+22] (with some extra work, complexity leveraging, and restrictions; see technical section). Note that combining the works of [BBDP22, OSY21] seems to implicitly yield rate-1 batch OT from DCR, and in turn correlated SPIR [BCM22]: 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.

Remark 6 (A note on the role of FSS in our template). Our template of section 3 for building $(N+1)$-party sublinear-communication MPC is built from N-party FSS with low-communication protocols for (N-party) share distribution and (2-party) oblivious share evaluation. While we believe phrasing our template using the abstraction of FSS makes it conceptually nicer, our results can be explained without the FSS machinery. The astute reader may observe that once the FSS scheme of section 5 is essentially "the trivial one" , up to how we drawn the line between the share generation and evaluation. Indeed, in lemma 23, we divide the tasks as:

1. In the (N-party) share distribution phase, parties $P_1, \ldots, P_N$ generate HSS shares.
2. In the i^{th} (2-party) oblivious evaluations, party P_i first locally evaluates its HSS share to obtain an additive share of the function's lookup table[7], then P_0 and P_i perform oblivious lookups with correlated SPIR.

If we push the local HSS evaluations to the share distribution phase, then the function secret-sharing scheme becomes essentially trivial, although the protocols for securely distributing the function shares and subsequently evaluating them obliviously remain highly non-trivial.

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 $C_{x_0} :=$

[7] For functions with polynomial-size lookup-tables, the "trivial" FSS scheme is the one which deals out additive shares of the lookup-tables. This can be extended to functions which can be computed by a loglog-depth circuits and layered circuits.

$C(x_0, \cdot, \ldots, \cdot)$. 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 Figure 1. Here, $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ represents an ideal N-party functionality for N-FSS share generation (defined formally in Figure 2 of Section 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 Figure 3 of Section 3), where P_i and P_0 respectively provide inputs k_i and x_0, and P_0 learns the evaluation $\mathsf{FSS.Eval}(i, k_i, x_0)$.

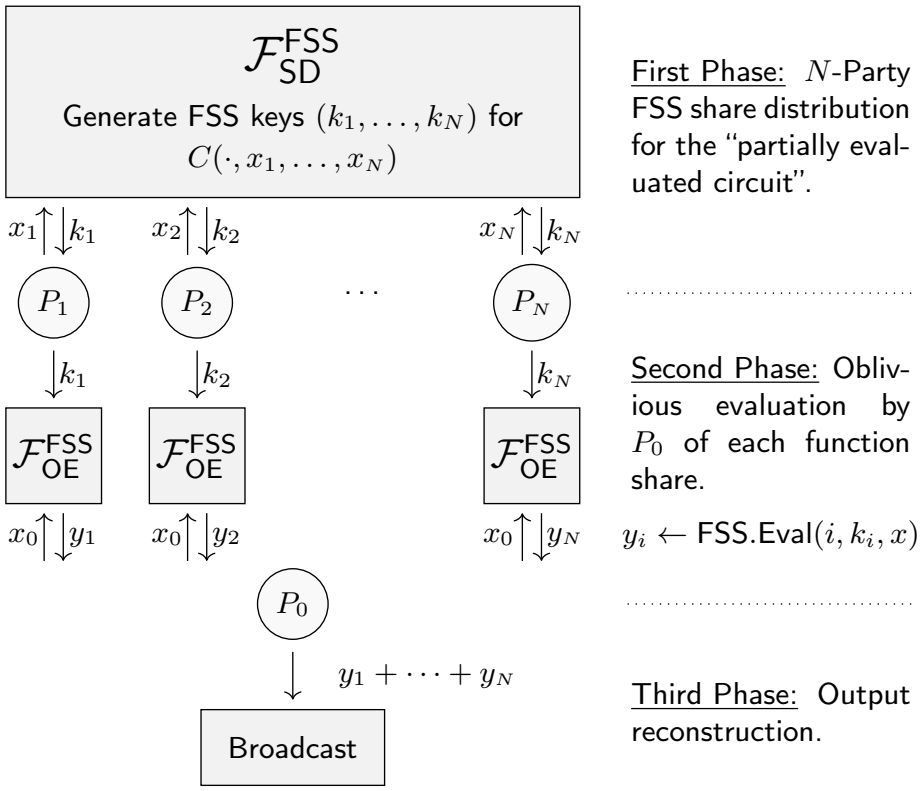

Fig. 1: Template for $(N+1)$-party sublinear secure computation of C from N-party additive FSS.

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 out-

put 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.[8]

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 [BGI17], 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. [COS+22] that for the specific DCR-based HSS construction of [OSY21, RS21], $\mathsf{HSS.Eval}$ 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 [BCM22], 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

[8] 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 Lemma 17.

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 [BGI16a] or from LWE with polynomial-size modulus [DHRW16, BKS19]. 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

2.1.1 Quadratic Residuosity Assumption (QR) We say that N is a Blum integer if $N = p \cdot q$ for some primes p and q such that $p \pmod 4 \equiv q \pmod 4 \equiv 3$. We denote by $\mathbb{J}_N$ the multiplicative group of the elements in $\mathbb{Z}_N^\star$ with Jacobi symbol $+1$ and by $\mathbb{QR}_N$ the multiplicative group of quadratic residues modulo N with generator g. Note that $\mathbb{QR}_N$ is a subgroup of $\mathbb{J}_N$, and that $\mathbb{QR}_N$ and $\mathbb{J}_N$ have order $\frac{\phi(N)}{4}$ and $\frac{\phi(N)}{2}$ respectively, where $\phi(\cdot)$ is Euler's totient function. It is useful to write $\mathbb{J}_N : \mathbb{H} \times \mathbb{QR}_N$, where $\mathbb{H}$ is the multiplicative group $(\pm 1, \cdot)$ of order 2. Note that is N is a Blum integer then $\gcd(2, \frac{\phi(N)}{4}) = 1$ and $-1 \in \mathbb{J}_N \setminus \mathbb{QR}_N$.

Definition 7 (Quadratic Residuosity Assumption, [GM82]). *Let N be a uniformly sampled Blum integer and let $\mathbb{QR}_N$ be the multiplicative group of quadratic residues modulo N with generator g. We say the* QR *assumption holds with respect to $\mathbb{QR}_N$ if for any p.p.t. adversary $\mathcal{A}$*

$$|\Pr_{a\xleftarrow{\$}\mathbb{QR}_N}[\mathcal{A}(N,g,a)=1]-\Pr_{a\xleftarrow{\$}\mathbb{QR}_N}[\mathcal{A}(N,g,(-1)\cdot a)=1]|\leq \mathrm{negl}(\lambda).$$

2.1.2 Learning With Errors (LWE)

Definition 8 (Decisional Learning with Errors). *Let $n\geq 1$ and $q\geq 2$ be integers. Let χ be an error distribution over $\mathbb{Z}$ and χ_{sk} be a secret key distribution over $\mathbb{Z}^n$. For $\vec{s}\xleftarrow{\$}\chi_{\mathsf{sk}}$, define $\mathsf{LWE}_{\chi,\vec{s}}$ to be the distribution obtained by sampling $\vec{a}\xleftarrow{\$}\mathbb{Z}_q^n$ uniformly at random, $\vec{e}\xleftarrow{\$}\chi$, and outputting $(\vec{a}, b =<\vec{a},s>+e)\in\mathbb{Z}_q^{n+1}$. The decisional-$\mathsf{LWE}_{n,q,\chi,\chi_{\mathsf{sk}}}$ problem asks to distinguish polynomially many samples $(\vec{a}_i,b_i)\xleftarrow{\$}\mathsf{LWE}_{\chi,\vec{s}}$ from the same number of samples taken from the uniform distribution on $(\mathbb{Z}_q^n,\mathbb{Z}_p)$, where the secret $\vec{s}$ is sampled accordng to $\chi_s k$.*

2.1.3 Learning Parity with Noise (LPN) Our constructions rely on the Learning Parity with Noise assumption [BFKL94] (LPN) over $\mathbb{F}_2$ (which is the most standard variant of LPN, but other fields can be considered), *a.k.a.* binary LPN. Unlike the LWE assumption, in LPN the noise is assumed to have a small Hamming weight. Concretely, the noise is 1 in a small fraction of the coordinates and 0 elsewhere. $\mathsf{Ber}_r(\mathbb{F}_2)$ denote the distribution which outputs 1 with probability r, and 0 with probability $1-r$.

Definition 9 (Learning Parity with Noise (LPN)). *For dimension $k=k(\lambda)$, number of samples (or block length) $q=q(\lambda)$, noise rate $r=r(\lambda)$, the $\mathbb{F}_2$-$\mathsf{LPN}(k,q,r)$ assumption states that*

$$\begin{aligned}&\{(A,\vec{b})\mid A\xleftarrow{\$}\mathbb{F}^{q\times k},\vec{e}\xleftarrow{\$}\mathsf{Ber}_r(\mathbb{F}_2)^q,\vec{s}\xleftarrow{\$}\mathbb{F}_2^k,\vec{b}\leftarrow A\cdot\vec{s}+\vec{e}\}\\ \overset{\mathrm{c}}{\approx}&\{(A,\vec{b})\mid A\xleftarrow{\$}\mathbb{F}_2^{q\times k},\vec{b}\xleftarrow{\$}\mathbb{F}_2^q\}\end{aligned}$$

Here and in the following, all parameters are functions of the security parameter λ and computational indistinguishability is defined with respect to λ.

Note that the search LPN problem, of finding the vector can be reduced to the decisional LPN assumption [BFKL94, AIK09].

2.1.4 Decisional Diffie-Hellman (DDH)

Definition 10 (Decisional Diffie-Hellman). *We say that the Decisional Diffie-Hellman assumption (*DDH*) holds if there exists a PPT group generator $\mathcal{IG}$ with the following properties. The output of $\mathcal{IG}(1^\lambda)$ is a pair $(\mathbb{G},g)$ where $\mathbb{G}$ describes a cyclic group of a prime order q (where we use multiplicative notations for the group operation) and g describes a group generator. We assume that q is included in the group description $\mathbb{G}$. We also assume the existence of an efficient algorithm that given $\mathbb{G}$ and descriptions of group elements h_1,h_2 outputs a description of h_1h_2. Finally, we require that for every nonuniform polynomial-time algorithm $\mathcal{A}$ there is a negligible function ϵ such that:*

$$\begin{aligned}|\Pr[\mathcal{A}(\mathbb{G},g,g^a,g^b,g^{ab})=1\colon(\mathbb{G},g)\xleftarrow{\$}\mathcal{IG};(a,b)\xleftarrow{\$}\mathbb{Z}_q^2]-&\\ \Pr[\mathcal{A}(\mathbb{G},g,g^a,g^b,g^c)=1\colon(\mathbb{G},g)\xleftarrow{\$}\mathcal{IG};(a,b,c)\xleftarrow{\$}\mathbb{Z}_q^3]|\leq\epsilon(\lambda).&\end{aligned}$$

2.1.5 Decision Composite Residuosity Assumption Let SampleModulus be a polynomial-time algorithm that on input the security parameter λ, outputs (N,p,q), where $N=pq$ for λ-bit primes p and q.

Definition 11 (Decision Composite Residuosity assumption, [Pai99]). *Let λ be the security parameter. We say that the Decision Composite Residuosity (*DCR*) problem is hard relative to* SampleModulus *if $(N,x)\overset{\mathrm{c}}{\approx}(N,x^N)$ where $(N,p,q)\xleftarrow{\$}\mathsf{SampleModulus}(1^\lambda)$, $x\xleftarrow{\$}\mathbb{Z}_{N^2}^*$, and x^N is computed modulo N^2.*

Note that $\mathbb{Z}_{N^2}^*$ can be written as a product of subgroups $\mathbb{H}\times\mathbb{NR}_N$, where $\mathbb{H}=\{(1+N)^i : i\in[N]\}$ is of order N, and $\mathbb{NR}_N=\{x^N : x\in\mathbb{Z}_{N^2}^*\}$ is the subgroup of N-th residues that has order $\phi(N)$.

2.2 Function Secret Sharing and Homomorphic Secret Sharing

We follow the function secret sharing definition of [BGI16b], for the specific leakage function which reveals the input and output domain sizes $(1^n, 1^m)$ of the secret function.

Definition 12 (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 $x \in \{0,1\}^n$,*

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

- ***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* $\mathsf{Sim}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^n, 1^m)$.

For consistency, we consider a definition of Homomorphic Secret Sharing with an analogous simulation-based security guarantee, with "leakage" corresponding to the input length n. This notion is equivalent to the natural indistinguishability-based definition (where simulation takes place by simply sharing a fixed input of required length).

Definition 13 (Homomorphic Secret Sharing). *An N-party* Homomorphic Secret-Sharing (HSS) *scheme (with additive reconstruction) for a class $\mathcal{F}$ of functions over a finite field $\mathbb{F}$ is a pair of algorithms* $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *with the following syntax and properties:*

- $\mathsf{Share}(1^\lambda, x)$*: On input 1^λ (the security parameter) and $x \in \mathbb{F}^{n(\lambda)}$ (the input), the* sharing algorithm Share *outputs N input shares $(x^{(1)}, \ldots, x^{(N)})$.*
- $\mathsf{Eval}(i, f, x^{(i)})$*: On input $i \in [N]$ (the party index), $f \in \mathcal{F}$ (the function to be homomorphically evaluated, implicitly assumed to specify input and output lengths n, m), and $x^{(i)}$ (the i^{th} input share), the* evaluation algorithm Eval *outputs the i^{th} output share $y^{(i)} \in \mathbb{F}^m$.*

- ***Correctness:*** *For any 1^λ, input $x \in \mathbb{F}^{n(\lambda)}$, and any function $f \in \mathcal{F}$,*

$$\Pr\left[y^{(1)} + \cdots + y^{(N)} = f(x) \ : \ \begin{array}{l}(x^{(1)}, \ldots, x^{(N)}) \xleftarrow{\$} \mathsf{HSS.Share}(1^\lambda, x) \\ y^{(i)} \xleftarrow{\$} \mathsf{HSS.Eval}(i, f, x^{(i)}),\ i = 1 \ldots N\end{array}\right] = 1 \ .$$

- ***Security:*** *For every set of corrupted parties $\mathcal{D} \subsetneq [N]$, there exists a probabilistic polynomial-time algorithm* $\mathsf{Sim}^{\mathsf{HSS}}$ *(a* simulator*), such that for every sequence of inputs $x_1, x_2, \cdots \in \mathbb{F}^{n(\lambda)}$ the outputs of the following experiments* $\mathsf{Real}^{\mathsf{HSS}}$ *and* $\mathsf{Ideal}^{\mathsf{HSS}}$ *are computationally indistinguishable:*
  - $\mathsf{Real}^{\mathsf{HSS}}(1^\lambda)$: $(x^{(1)}, \ldots, x^{(N)}) \xleftarrow{\$} \mathsf{HSS.Share}(1^\lambda, x_\lambda)$*; Output* $(x^{(i)})_{i \in \mathcal{D}}$.
  - $\mathsf{Ideal}^{\mathsf{HSS}}(1^\lambda)$: *Output* $\mathsf{Sim}^{\mathsf{HSS}}(1^\lambda, 1^N, 1^n)$.

Remark 14 (Compact Single-Function HSS). A *single-function HSS* is an HSS scheme for a singleton function class. Let $\mathcal{F}$ be a (not necessarily singleton) function class. We say there exists *compact single-function HSS for any function in $\mathcal{C}$*, if for every $f\colon \mathbb{F}^n \to \mathbb{F}^m \in \mathcal{F}$ there exists an HSS scheme HSS_f for $\{f\}$ such that the circuit-size of $\mathsf{HSS}_f.\mathsf{Share}$ is a fixed polynomial in n (and otherwise independent of f).

This notion can be seen as a weakening of compact HSS for $\mathcal{C}$ where the function to be homomorphically evaluated is known when running the sharing algorithm.

Definition 15 (Las Vegas HSS). *A* Las Vegas *N-party homomorphic secret-sharing scheme with additive reconstruction is defined as above, with the following modification:*

1. *The algorithm* Eval *takes as input a* failure bound *δ, and additonnally outputs a confidence flag* $\mathsf{flag}_b \in \{\bot, \top\}$ *to indicate full confidence ($\top$) or a possibility of failure ($\bot$).* Eval *can run in time polynomial in its input length and in $1/\delta$.*
2. *The correctness notion is relaxed to the following notion of* Las Vegas correctness*: for every input $x \in \mathbb{F}^n$, function $f \in \mathcal{F}$ with input length n, and failure bound $\delta > 0$, we have:*

$$\Pr\left[\exists i \leq N, (\mathsf{flag}_i = \bot)\right] \leq \delta,$$
$$\textit{and } \Pr[(\exists i \leq N, (\mathsf{flag}_i = \top) \wedge (\oplus_{i \leq n} y^{(i)} \neq f(x))] = 0,$$

where the probability is taken over the coins of Gen *and* $\mathsf{Eval}(\cdot, \cdot, \cdot, \delta)$*. We implicitly assume each execution of* Eval *to take an additional nonce input, which enables different invocations to have (pseudo)-independent failure probabilities. (See [BGI16a] for more a long-form discussion.)*

2.3 Universal Composability

We refer the reader to [Can01] for details on the universal composability framework. The framework is based on the real/ideal paradigm for arguing about the security of a protocol. We say that a protocol π $\mathsf{UC\text{-}realises}$ $(\mathsf{with\ computational\ security})$ an ideal functionality $\mathcal{F}$ in the presence of static semi-honest adversary corrupting at most t parties, if for any *p.p.t.* static semi-honest t-adversary $\mathcal{A}$ and any *p.p.t.* environment $\mathcal{Z}$, there exists a p.p.t. ideal-model t-adversary Sim such that the output distribution of $\mathcal{Z}$ in the ideal-model computation of $\mathcal{F}$ with Sim is *computationally indistinguishable* from its output distribution in the real-model execution of π with $\mathcal{A}$. The composition theorem of [Can01] states the following.

Theorem 16 (Composition Theorem [Can01], informal). *Let ρ be a protocol that UC-realises $\mathcal{F}$ in the presence of adaptive semi-honest t-adversaries, and let π be a protocol that UC-realises $\mathcal{G}$ in the $\mathcal{F}$-hybrid model in the presence of adaptive semi-honest t-adversaries. Then, for any* p.p.t. *adaptive semi-honest t-adversary $\mathcal{A}$ and any* p.p.t. *environment $\mathcal{Z}$, there exists a* p.p.t. *adaptive semi-honest t-adversary* Sim *in the $\mathcal{F}$-hybrid model such that the output distribution of $\mathcal{Z}$ when interacting with the protocol π and* Sim *is computationally indistinguishable from its output distribution when interacting with the protocol π^ρ (where every call to $\mathcal{F}$ is replaced by an execution of ρ) and $\mathcal{A}$ in the real model.*

2.4 Notations

Throughout this paper, N denotes a number of parties (but the total number of parties is sometimes $N+1$). The number of inputs and outputs of an arithmetic circuit are denoted n and m respectively. If f is a function, $\tilde{f}$ is used to describe a polynomial-size description of f.

By default, vectors are assumed to be column vectors. If $\vec{x}$ and $\vec{y}$ are two (column) vectors, we use $\vec{x}||\vec{y}$ to denote the (column) vector obtained by their concatenation. We write $\vec{x} \otimes \vec{y}$ to denote the tensor product between $\vec{x}$ and $\vec{y}$, i.e., the vector of length $n_x n_y$ with coordinates $x_i y_j$ (where n_x is the length of $\vec{x}$ and n_y is the length of $\vec{y}$). We write $\vec{x}^{\otimes 2}$ for $\vec{x} \otimes \vec{x}$, and more generally, $\vec{x}^{\otimes k}$ for the *k-th tensor power* of $\vec{x}$, $\vec{x} \otimes \vec{x} \otimes \cdots \otimes \vec{x}$ (k times). Observe that evaluating a degree-d n-variate polynomial on some size-n vector $\vec{x}$ can be done by computing the inner product of the vector of coefficients of the polynomial with $\vec{x}^{\otimes d} \| \ldots \| \vec{x}^{\otimes 1} \| 1)$. Indeed, each $\vec{x}^{\otimes k}$ corresponds the vector of every degree-k monomial in the coordinates of $\vec{x}$.

If $k \geq 1$ is an integer, $[k]$ denotes the set $\{1, \ldots, k\}$, $[0, k]$ denotes the set $\{0, 1, \ldots, k\}$, and $\mathfrak{S}_k$ denotes the symmetric group of order k, *i.e.* the set of all permutations on $[k]$. If A and B are sets, A^B denotes the set of all functions from A to B.

We say that a function $\mathrm{negl}\colon \mathbb{N} \to \mathbb{R}^+$ is *negligible* if it vanishes faster than every inverse polynomial. For two families of distributions $X = \{X_\lambda\}$ and $Y = \{Y_\lambda\}$ indexed by a security parameter $\lambda \in \mathbb{N}$, we write $X \overset{\mathrm{c}}{\approx} Y$ if X and Y are *computationally indistinguishable* (*i.e.* any family of circuits of size $\mathsf{poly}(\lambda)$ has a negligible distinguishing advantage), $X \overset{\mathrm{s}}{\approx} Y$ if they are *statistically indistinguishable* (*i.e.* the above holds for arbitrary, unbounded, distinguishers), and $X \equiv Y$ if the two families are identically distributed.

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 18 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 lemma 17 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 $f(x)$.

Lemma 17. *Let* $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ *be an N-party additive function secret sharing scheme for some function family $\mathcal{F}$, and let* $\mathsf{Sim}^{\mathsf{FSS}}$ *be a simulator defined as in definition 12. Then, the following holds:*

Reconstruction-Only (given some keys, all shares, the input, and auxiliary information): *Let* $\mathsf{info}\colon \{0,1\}^\star \to \{0,1\}^\star$ *be any function which, on input the description $\tilde{f}$ of some function $f \in \mathcal{F}$, outputs some partial information* $\mathsf{info}(\tilde{f})$ *about f. For every set of corrupted parties $\mathcal{D} \subsetneq [N]$, there exists a probabilistic polynomial-time algorithm* $\mathsf{Sim}_{\mathsf{rec,info}}^{\mathsf{FSS}}$ *(a* simulator*), such that for every sequence $\tilde{f}_1, \tilde{f}_2, \ldots$ of polynomial-size function description of functions $f_1, f_2, \cdots \in \mathcal{F}$, for every sequence of inputs $(x_\lambda)_{\lambda\in\mathbb{N}^\star} \in D_{f_1} \times D_{f_2} \times \ldots$, the outputs of the following experiments* $\mathsf{Real}_{\mathsf{rec,info}}$ *and* $\mathsf{Ideal}_{\mathsf{rec,info}}$ *are computationally indistinguishable:*

- $\mathsf{Real}_{\mathsf{rec,info}}(1^\lambda)$:
  - $(k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda)$;
  - $y_i \leftarrow \mathsf{Eval}(i, k_i, x_\lambda)$ *for* $i \notin \mathcal{D}$;
  - *Output* $(x_\lambda, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\notin\mathcal{D}}, \mathsf{info}(\tilde{f}_\lambda))$.
- $\mathsf{Ideal}_{\mathsf{rec,info}}(1^\lambda)$: *Output* $\mathsf{Sim}_{\mathsf{rec,info}}(1^\lambda, 1^n, 1^m, x_\lambda, f_\lambda(x_\lambda), \mathsf{info}(\tilde{f}_\lambda))$.

Proof. We first consider the intermediary step of "Function Secrecy (given some keys, and auxiliary information)".

1. **Function Secrecy (given some keys, and auxiliary information):** Let $\mathsf{info}\colon \{0,1\}^\star \to \{0,1\}^\star$ be any function which, on input the description $\tilde{f}$ of some function $f \in \mathcal{F}$, outputs some partial information $\mathsf{info}(\tilde{f})$ about f. For every set of corrupted parties $\mathcal{D} \subsetneq [N]$, there exists a probabilistic polynomial-time algorithm $\mathsf{Sim}_{\mathsf{info}}^{\mathsf{FSS}}$ (a *simulator*), such that for every sequence $\tilde{f}_1, \tilde{f}_2, \ldots$ of polynomial-size function description of functions $f_1, f_2, \cdots \in \mathcal{F}$ (we denote $n(\lambda)$ and $m(\lambda)$ respectively the input and output length of f_λ for $\lambda \in \mathbb{N}^\star$), the outputs of the following experiments $\mathsf{Real}_{\mathsf{info}}$ and $\mathsf{Ideal}_{\mathsf{info}}$ are computationally indistinguishable:
   - $\mathsf{Real}_{\mathsf{info}}(1^\lambda)$: $(k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda)$; Output $((k_i)_{i\in\mathcal{D}}, \mathsf{info}(\tilde{f}_\lambda))$.
   - $\mathsf{Ideal}_{\mathsf{info}}(1^\lambda)$: Output $\mathsf{Sim}_{\mathsf{info}}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^{n(\lambda)}, 1^{m(\lambda)}, \mathsf{info}(\tilde{f}_\lambda))$.

   And furthermore, such a simulator can be built as:

   $$\mathsf{Sim}_{\mathsf{info}}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^{n(\lambda)}, 1^{m(\lambda)}, \mathsf{info}(\tilde{f}_\lambda))\colon \text{Output } (\mathsf{Sim}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^{n(\lambda)}, 1^{m(\lambda)}), \mathsf{info}(\tilde{f}_\lambda)).$$

   The above is true because the notion of FSS security is hereditary in that any FSS scheme for some function class $\mathcal{F}$ is also an FSS scheme for any subclass $\mathcal{F}' \subseteq \mathcal{F}$, and furthermore the same simulator can be used. In particular this is true for $\mathcal{F}'$ obtained by quotienting $\mathcal{F}$ by the auxiliary information given to an adversary.

2. **Reconstruction-Only (given some keys, all shares, and the input):** We now prove the lemma. Let $\mathsf{info}\colon \{0,1\}^\star \to \{0,1\}^\star$ be any function which, on input the description $\tilde{f}$ of some function $f \in \mathcal{F}$, outputs some partial information $\mathsf{info}(\tilde{f})$ about f. Let info' be the function which, on input $\tilde{f}_\lambda$, outputs $\mathsf{info}'(\tilde{f}_\lambda) \coloneqq (x_\lambda, f_\lambda(x_\lambda))$. Let $\mathcal{D} \subsetneq [N]$, and let $\mathcal{D}^\star$ be any size-$(N-1)$ superset of $\mathcal{D}$ in $[N]$ (*i.e.* $\mathcal{D} \subseteq \mathcal{D}^\star \subsetneq [N]$, $|\mathcal{D}^\star| = N-1$). Denote u the unique element of $[N] \setminus \mathcal{D}^\star$. Let $(f_\lambda)_{\lambda\in\mathbb{N}^\star} \in \mathcal{F}^{\mathbb{N}^\star}$ and $(x_\lambda)_{\lambda\in\mathbb{N}^\star} \in (D_{f_1} \times D_{f_2} \times \dots)$. By function secrecy (given some keys, and auxiliary information), there exists a simulator $\mathsf{Sim}^{\mathsf{FSS}}_{(\mathsf{info}',\mathsf{info})}$ such that the following distributions Δ_1 and Δ_2 are computationally indistinguishable:

$$\Delta_1 \coloneqq \left\{((k_i)_{i\in\mathcal{D}^\star}, \mathsf{info}'(\tilde{f}_\lambda), \mathsf{info}(\tilde{f}_\lambda)) : (k_1,\dots,k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda)\right\}$$
$$\Delta_2 \coloneqq \left\{\mathsf{Sim}^{\mathsf{FSS}}_{(\mathsf{info}',\mathsf{info})}(1^\lambda, 1^N, 1^{n(\lambda)}, 1^{m(\lambda)}, \mathsf{info}'(\tilde{f}_\lambda), \mathsf{info}(\tilde{f}_\lambda))\right\}.$$

Now consider the distributions Δ_3, Δ'_3, Δ_4, and Δ'_4 defined as:

$$\Delta_3 \coloneqq \left\{(x_\lambda, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\in\mathcal{D}^\star\setminus\mathcal{D}}, \mathsf{info}(\tilde{f}_\lambda)) : \begin{array}{l}(k_1,\dots,k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda)\\ y_i \leftarrow \mathsf{Eval}(i, k_i, x_\lambda), i \in \mathcal{D}^\star \setminus \mathcal{D}\end{array}\right\}$$
$$\Delta'_3 \coloneqq \left\{(x_\lambda, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\notin\mathcal{D}}, \mathsf{info}(\tilde{f}_\lambda)) : \begin{array}{l}(k_1,\dots,k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda)\\ y_i \leftarrow \mathsf{Eval}(i, k_i, x_\lambda), \text{ for } i \notin \mathcal{D}\end{array}\right\}$$

$$\Delta_4 \coloneqq \left\{(\alpha, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\in\mathcal{D}^\star\setminus\mathcal{D}}, \gamma) : \begin{array}{l}((k_i)_{i\in\mathcal{D}^\star}, (\alpha,\beta), \gamma) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{FSS}}_{(\mathsf{info}',\mathsf{info})}\\ y_i \leftarrow \mathsf{Eval}(i, k_i, \alpha), \text{ for } i \in \mathcal{D}^\star \setminus \mathcal{D}\end{array}\right\}$$
$$\Delta'_4 \coloneqq \left\{(\alpha, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\notin\mathcal{D}}, \gamma) : \begin{array}{l}((k_i)_{i\in\mathcal{D}^\star}, (\alpha,\beta), \gamma) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{FSS}}_{(\mathsf{info}',\mathsf{info})}\\ y_i \leftarrow \mathsf{Eval}(i, k_i, \alpha), \text{ for } i \in \mathcal{D}^\star\\ y_u \leftarrow \beta - \sum_{i\in\mathcal{D}^\star} y_i\end{array}\right\}.$$

Note that the only difference between Δ_3 and Δ'_3 (*resp.* Δ_4 and Δ'_4) is whether or not y_u is part of the output of the distribution (where $\{u\} = [N] \setminus \mathcal{D}^\star$). Because $\Delta_1 \overset{\mathrm{c}}{\approx} \Delta_2$, the following algorithm cannot distinguish between Δ_1 and Δ_2: On input $((k_i)_{i\in\mathcal{D}^\star}, \mathsf{info}(\tilde{f}_\lambda) = (\alpha,\beta))$, set $y_i \leftarrow \mathsf{Eval}(i, k_i, \alpha)$ for $i \in \mathcal{D}^\star \setminus \mathcal{D}$, then output $(\alpha, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\in\mathcal{D}^\star\setminus\mathcal{D}})$. This implies, by perfect correctness of the additive FSS scheme, that $\Delta_3 \overset{\mathrm{c}}{\approx} \Delta_4$, which in turn implies that the following algorithm cannot distinguish between Δ_3 and Δ_4: On input$(\alpha, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\in\mathcal{D}^\star\setminus\mathcal{D}})$, set $y_u \leftarrow \beta - \sum_{i\in\mathcal{D}^\star} y_i$, and output $(\alpha, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\notin\mathcal{D}})$. Therefore, $\Delta'_3 \overset{\mathrm{c}}{\approx} \Delta'_4$.

□

3.2 The Secure Computation Protocol

We define the ideal functionalities $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}$ (fig. 2) for N-party FSS share distribution, and $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{OE}}$ (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 $(\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}, \mathcal{F}^{\mathsf{FSS}}_{\mathsf{OE}})$-hybrid model.

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

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

$\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}$ interacts with the N parties $P_1,\dots,P_N$ 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 $(k_1,\dots,k_N) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, \tilde{f}_{x_1,\dots,x_N})$, where $\tilde{f}_{x_1,\dots,x_N}$ is a description of $f_{x_1,\dots,x_N}$; Output k_i to each party P_i (for $1 \le i \le N$).

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

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 $\vec{x} \leftarrow x_0 \| x_1 \| \ldots \| x_N$.

Output: Compute $\vec{y} \leftarrow C(\vec{x})$; Output $\vec{y}$ 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.

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 $(\texttt{input}, i, x_i)$ 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 $(\mathsf{sid}_i, \texttt{Client}, x_0)$ to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C)$ and P_i sends $(\mathsf{sid}_i, \texttt{Server}, i, k_i)$ 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 $\vec{y} \leftarrow y_1 + \cdots y_N$, and sends $\vec{y}$ to all parties.
4. Every party outputs $\vec{y}$.

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

Theorem 18 (Template for $(N+1)$-Party Sublinear MPC from N-Party FSS). *Let $N \geq 2$. 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":*

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

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 $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C)$.

Proof. Let $\mathcal{A}$ be a semi-honest, static adversary that interacts with parties $P_0, P_1, \ldots, P_N$ running protocol Π_C in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid model. We construct in fig. 6 a simulator Sim such that no environment $\mathcal{Z}$ can distinguish with non-negligible probability whether it is interacting with $\mathcal{A}$ and Π_C in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid model, or with Sim and $\mathcal{F}_{\mathsf{SFE}}(C)$.

Simulator Sim

Let $\mathcal{D} \subsetneq [0, N]$ be the subset of statically corrupted parties, and let $\mathcal{H} \coloneqq [0, N] \setminus \mathcal{D}$ be the (non-empty) set of honest parties. Let $(x_i)_{i \in \mathcal{D}}$ be the set of inputs of the semi-honestly corrupt parties.

Requires: Simulator $\mathsf{Sim}^{\mathsf{FSS}}$ is defined as in definition 12. If P_0 is corrupted (*i.e.* $0 \in \mathcal{D}$), then let function info be defined on $\mathcal{C}$ as $\mathsf{info}(g_{\alpha_1,\ldots,\alpha_N}) \coloneqq (\alpha_i)_{i \in \mathcal{D} \setminus \{0\}}$, and let simulator $\mathsf{Sim}_{\mathsf{rec,info}}^{\mathsf{FSS}}$ be defined as in lemma 17, so that:

$$\left\{ (x_\lambda, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\notin\mathcal{D}}, \mathsf{info}(\tilde{f}_\lambda)) \,:\, \begin{array}{l} (k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda) \\ y_i \leftarrow \mathsf{Eval}(i, k_i, x_\lambda), \text{ for } i \notin \mathcal{D} \end{array} \right\} \overset{\mathrm{c}}{\approx} \left\{ \mathsf{Sim}_{\mathsf{rec,info}}(1^\lambda, 1^n, 1^m, x_\lambda, f_\lambda(x_\lambda), \mathsf{info}(\tilde{f}_\lambda)) \right\} .$$

Simulation:

- **Simulating the communication with $\mathcal{Z}$:** Every input value that Sim receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment), and every output value written by $\mathcal{A}$ on its output tape is copied to Sim's own output tape (to be read by $\mathcal{Z}$).

- **Simulating the protocol's execution:**
  1. Sim activates $\mathcal{A}$ and receives the messages $(\mathtt{input}, i, x_i)_{i\in\mathcal{D}\setminus\{0\}}$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}(C)$ (on behalf of the corrupted parties) in a real execution.
  2. If P_0 is honest, Sim sets

$$(k_i)_{i\in\mathcal{D}} \xleftarrow{\$} \begin{cases} \mathsf{Sim}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^{\ell_0}, 1^m) & \text{if } \mathcal{D} \neq [1, N] \\ \mathsf{FSS.Gen}(1^\lambda, g_{x_1,\ldots,x_N}) & \text{if } \mathcal{D} = [1, N] . \end{cases}$$

     If P_0 is corrupted, Sim instead sets

$$(-, (k_i)_{i\in\mathcal{D}\setminus\{0\}}, (y_i)_{i\in[1,N]\setminus\mathcal{D}}, -) \xleftarrow{\$} \mathsf{Sim}_{\mathsf{rec,info}}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^{\ell_0}, 1^m, x_0, y, (x_i)_{i\in\mathcal{D}\setminus\{0\}}) .$$

     In either case, Sim then writes $(k_i)_{i\in\mathcal{D}\setminus\{0\}}$ on $\mathcal{A}$'s input tape (as if $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}(C)$ sent k_i to P_i for each $i \in \mathcal{D} \setminus \{0\}$).
  3. If P_0 is corrupted:
     Sim waits to receive the messages $(\mathsf{sid}_i, \mathtt{Client}, x_0)_{i\in\mathcal{D}\setminus\{0\}}$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C)$ (on behalf of the P_0).
  4. Sim sends $(\mathtt{input}, i, x_i)_{i\in\mathcal{D}}$ to $\mathcal{F}_{\mathsf{SFE}}(C)$ and waits to receive y (on behalf of the corrupted parties).

5. If P_0 is corrupted:
   For $i \notin \mathcal{D}$, Sim writes (sid_i, y_i) (recall that y_i was defined in step 2. above) on $\mathcal{A}$'s input tape (as if P_i sent it to P_0).
6. If P_0 is not corrupted then, for $i \in \mathcal{D}$, Sim writes y on $\mathcal{A}$'s input tape (as if P_0 sent it to P_i).

Fig. 6: Simulator Sim for an execution of protocol Π_C.

In order to prove that no environment $\mathcal{Z}$ can distinguish between the real and ideal worlds, we proceed by a case analysis over the set of corrupted parties $\mathcal{D}$.

- **If only P_0 is honest (*i.e.* $\mathcal{D} = [1, N]$):** In this case the simulation is perfect. Indeed, the simulator knows the corrupt parties' inputs $x_1, \ldots, x_N$, which allows it to perfectly emulate the ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}(C)$. The only other incoming messages received by the corrupted parties is the broadcast output, which is identical both in the real and ideal worlds by (perfect) correctness of the FSS scheme.
- **If P_0 and at least one other P_i are honest (*i.e.* $\mathcal{D} \subsetneq [1, N]$):** The only difference between the real and the ideal worlds is whether the corrupted parties receive real FSS keys or simulated ones. Indeed, by (perfect) correctness of the FSS the broadcast message received is identical in the real and the ideal world. By the first part of lemma 17 (function secrecy given some keys, and auxiliary information), the joint views of $\mathcal{Z}$ and $\mathcal{A}$ are indistinguishable in the real and the ideal worlds: even given the auxiliary information $(y, (x_i)_{i\in\mathcal{D}})$ about the function, it cannot distinguish the real corrupted keys $(k_i)_{i\in\mathcal{D}}$ from $(k_i)_{i\in\mathcal{D}} \xleftarrow{\$} \mathsf{Sim}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^{\ell_0}, 1^m)$.
- **If P_0 is corrupt (*i.e.* $\mathcal{D} \not\subseteq [1, N]$):** Note that since the adversary is only allowed to corrupt up to all but one parties, if P_0 is corrupt then at least one P_i ($i \in [1, N]$) is honest. The only difference between the real and the ideal world is whether the corrupted keys $(k_i)_{i\in\mathcal{D}\setminus\{0\}}$ and the honest shares $(y_i)_{i\in[1,N]\setminus\mathcal{D}}$ are real or simulated by $\mathsf{Sim}_{\mathsf{rec,info}}^{\mathsf{FSS}}$. It follows from the second part of lemma 17 (reconstruction-only given some keys, all shares, and auxiliary information) that the joint views of $\mathcal{Z}$ and $\mathcal{A}$ are indistinguishable in the real and the ideal worlds.

□

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 [BCM22] (and which can be instantiated from standard assumptions using the rate-1 batch OT from [BBDP22]).

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[9] 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

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

$d = \log\log n = \log\log\lambda + \mathcal{O}(1)$.

We extend in definition 19 the above notion of "loglog-depth" circuits to "loglog-depth" FSS schemes.

Definition 19 (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" circuits, which in particular are "log-local" because each gate has fan-in at most two, 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 emphasize that a loglog-depth circuit or FSS scheme is always log-local, but that the converse is not necessarily true if $\mathbb{F} \neq \mathbb{F}_2$.
We recall in definition 20 the notion of locality for circuits, and extend it to FSS schemes. We emphasize that a loglog-depth circuit or FSS scheme is always log-local, but that the converse is not necessarily true if $\mathbb{F} \neq \mathbb{F}_2$.

Definition 20 (Log-locality). *Let $\mathcal{F}$ be a class of functions with n inputs and m outputs over a finite field $\mathbb{F}$, and let $S_1, \ldots, S_m \subseteq [n]$.*

- ***Locality of functions.*** *We say a function $f \in \mathcal{F}$ is* $(S_1, \ldots, S_m)$-local *if, for every $j \in [m]$, the j^{th} output of f can be expressed as a function of the inputs indexed by S_j. If there exists a such a tuple $(S_1, \ldots, S_m)$ such that furthermore $\forall i \in [1, m], |S_i| \leq (\log n / \log\log n)$, we say that f is* log-local.
- ***Locality of function classes.*** *We say that $\mathcal{F}$ is $(S_1, \ldots, S_m)$-local if every function $f \in \mathcal{F}$ is. If there exists a such a tuple $(S_1, \ldots, S_m)$ such that furthermore $\forall i \in [1, m], |S_i| \leq (\log n / \log\log n)$, we say that $\mathcal{F}$ is* log-local.
- ***Locality of FSS schemes.*** *We say that an additive[10] FSS scheme for $\mathcal{F}$ is* $(S_1, \ldots, S_m)$-local *(*resp. *log-local) if its evaluation algorithm is $(S_1, \ldots, S_m)$-local (*resp. *log-local) in its last input. In other words, the $\mathsf{FSS.Eval}$ algorithm can be decomposed as deterministic algorithms $\mathsf{Eval}_1, \ldots, \mathsf{Eval}_m$ for which:*

$$\mathsf{FSS.Eval}(i, k_i, x) = (\mathsf{Eval}_1(i, k_i, x[S_1]), \ldots, \mathsf{Eval}_m(i, k_i, x[S_m]).$$

That is, each Eval_j takes as input the corresponding subset of bits $x[S_j]$ of the input x, and outputs the jth bit of $\mathsf{FSS.Eval}$.

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.

4.2.1 Correlated PIR. We recall the ideal functionality for *batch SPIR with correlated "mix and match" queries* ($\mathcal{F}_{\mathsf{corrSPIR}}$, fig. 7) from [BCM22], which we extend from the boolean to the arithmetic setting as *batch Oblivious (Multivariate) Polynomial Evaluation with correlated "mix and match" queries* ($\mathcal{F}_{\mathsf{corrOPE}}$, fig. 8).
In the boolean world, this corresponds to a batched form of SPIR, querying into k size-N databases,

[10] This notion of locality could be extended beyond additive FSS schemes.

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 $\vec{\alpha} = (\alpha_1, \ldots, \alpha_w)$ 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".

Definition 21 ("Mix and Match" Functions, adapted from [BCM22]). *A "mix and match" function* $\mathsf{MixAndMatch}\colon \mathbb{F}^w \to (\mathbb{F}^{N_{\mathsf{var}}})^k$ *is one parameterised by* k *ordered subsets of* N_{var} *elements of* $[w]$, $S_j = (s_{j,1}, \ldots, s_{j,n}) \in [w]^{N_{\mathsf{var}}}$ *for* $j \in [k]$ *such that:*

$$\forall \vec{\alpha} = (\alpha_1, \ldots, \alpha_w) \in \mathbb{F}^w, \mathsf{MixAndMatch}(\alpha_1, \ldots, \alpha_w) \coloneqq (x_1, \ldots, x_k),$$
$$\text{with } x_j \coloneqq \alpha_{s_{j,1}} \cdots \alpha_{s_{j,n}} \in (\mathbb{F}^{N_{\mathsf{var}}}).$$

Such a function is associated with an occurrence function, which counts the occurrences of each input position in the outputs:

$$\begin{aligned} t.\colon [w] &\to [k] \\ i \ &\mapsto t_i = \sum_{j=1}^{k} \mathbf{1}_{i \in S_j} \end{aligned}$$

Each t_i *(*$i \in [w]$*) can be decomposed as* $t_i = t_{i,1} + \cdots + t_{i,N_{\mathsf{var}}}$*, where* $t_{i,j'}$ *is equal to the number of values of* $j \in [k]$ *such that* $s_{j,j'} = i$*.*

- $\mathsf{MixAndMatch}$ *is said to be* T-balanced *if* $\forall i \in [w], \forall j' \in [N_{\mathsf{var}}], t_{i,j'} \leq T$*.*
- $\mathsf{MixAndMatch}$ *is said to be* T-balanceable *if it can be expressed as the function* $\mathsf{MixAndMatch} = (\mathsf{MixAndMatch}' \circ \mathsf{replicate})$*, where* $\mathsf{MixAndMatch}\colon \mathbb{F}^{w'} \to (\mathbb{F}^{N_{\mathsf{var}}})^k$ *is a* T*-balanced mix-and-match function and* replicate *is defined as:*

$$\begin{aligned} \mathsf{replicate}\colon \quad \mathbb{F}^w &\to \mathbb{F}^{w'} \\ (b_1, \ldots, b_w) &\mapsto (b_1^{\|\lceil t_1/T \rceil} \| \ldots \| b_w^{\|\lceil t_w/T \rceil}) \end{aligned} \qquad \text{where } w' \coloneqq \sum_{i \in [w]} \lceil t_i/T \rceil.$$

Functionality $\mathcal{F}_{\mathsf{corrSPIR}}$

The functionality $\mathcal{F}_{\mathsf{corrSPIR}}$ is parameterised by the number k of SPIRs in the batch, the size N of each database, and the number w of selection bits. Furthermore, it is parameterised by a public T-balanceable "mix and match" function (definition 21) $\mathsf{MixAndMatch}\colon \{0,1\}^w \to [N]^k$. $\mathcal{F}_{\mathsf{corrSPIR}}$ interacts with an ideal sender $\mathbf{S}$ and an ideal receiver $\mathbf{R}$ via the following queries.

1. On input $(\texttt{sender}, \vec{M} = (\vec{m}_i)_{i \in [k]})$ from $\mathbf{S}$ (where $\vec{m}_i = (m_{i,j})_{j \in [N]} \in \{0,1\}^N$), store $\vec{M}$.
2. On input $(\texttt{receiver}, (\alpha_j)_{j \in [w]})$ from $\mathbf{R}$ (where $(\alpha_j)_{j \in [w]} \in \{0,1\}^w$), check if a tuple of inputs $\vec{M}$ has already been recorded; if so, compute $(x_1, \ldots, x_k) \coloneqq \mathsf{MixAndMatch}(\alpha_1, \ldots, \alpha_w) \in [N]^k$, send $(m_{i,x_i})_{i \in [k]}$ to $\mathbf{R}$, and halt.

If the functionality receives an incorrectly formatted input, it aborts.

Fig. 7: Ideal Functionality $\mathcal{F}_{\mathsf{corrSPIR}}$ for Batch SPIR with Correlated "Mix and Match" Queries

Functionality $\mathcal{F}_{\mathsf{corrOPE}}$

The functionality $\mathcal{F}_{\mathsf{corrOPE}}$ is parameterised by a finite field $\mathbb{F}$, the number k of oblivious (multivariate) polynomial evaluations (OPE) in the batch, the number N_{var} of variables of each polynomial, the degree d of each polynomial, and the size w of the joint inputs vector. Let $\mathsf{nb}_{N_{\mathsf{var}},d} \colon \sum_{d'=0}^{d} \binom{N_{\mathsf{var}}+d'}{N_{\mathsf{var}}}$ denote the maximum number of monomials of a degree-d N_{var}-variate polynomial. Furthermore, it is parameterised by a public polylog-balanced "mix and match" function (definition 21) $\mathsf{MixAndMatch} \coloneqq \mathbb{F}^w \to \mathbb{F}^k$. $\mathcal{F}_{\mathsf{corrOPE}}$ interacts with an ideal sender $\mathbf{S}$ and an ideal receiver $\mathbf{R}$ via the following queries.

1. On input $(\texttt{sender}, \vec{c}_i)$ from $\mathbf{S}$ (where $(\vec{c}_i)_{i\in[k]} \in \mathbb{F}^{k\cdot \mathsf{nb}_{N_{\mathsf{var}},d}}$), store $(\vec{c}_i)_{i\in[k]}$.
   // $\vec{c}_i$ is the vector of coefficients of the i^{th} polynomial (with zeroes).
2. On input $(\texttt{receiver}, (\alpha_j)_{j\in[w]})$ from $\mathbf{R}$ (where $(\alpha_j)_{j\in[w]} \in \mathbb{F}^w$), check if a tuple of inputs $(\vec{c}_i)_{i\in[k]}$ has already been recorded; if so, compute $(x_1, \dots, x_k) \coloneqq \mathsf{MixAndMatch}(\alpha_1, \dots, \alpha_w) \in (\mathbb{F}^{N_{\mathsf{var}}})^k$, compute $y_i \leftarrow \vec{c}_i \cdot (x_i^{\otimes d} \| \dots \| x_i^{\otimes 1} \| 1) \in \mathbb{F}$ for $i \in [k]$, send $(y_i)_{i\in[k]}$ to $\mathbf{R}$, and halt.
   // See section 2.4 for the meaning of $\otimes$. Each x_i is the vector of the N_{var} inputs to the i^{th} polynomial in the batch, whose coefficients are given by $\vec{c}_i$. Therefore, y_i is the evaluation of the i^{th} polynomial on input x_i.

If the functionality receives an incorrectly formatted input, it aborts.

Fig. 8: Ideal Functionality $\mathcal{F}_{\mathsf{corrOPE}}$ for Batch Oblivious Polynomial Evaluation with Correlated "Mix and Match" Inputs

4.2.2 Oblivious Evaluation of LogLog-Depth FSS from PIR. Let $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ be a loglog-depth, $(S_1, \dots, S_m)$-local FSS scheme (definition 19). 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. 9.
Note that this notion of corrOPE, as defined in fig. 8, 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 19) with $n = \ell_0 + \ell_1 + \dots + \ell_N$ 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 $[n]$:

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

  We assume that each of the S_i is ordered in such a way that the function MixAndMatch associated with $(S_1, \dots, S_m)$ is polylog-balanced[a] (definition 21).
- $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ is an $(S_1, \dots, S_m)$-local (additive) FSS scheme for $\mathcal{C}$, whose local evaluation algorithms 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 MixAndMatch, and in turn 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 $(\texttt{sender}, (\vec{c}_j)_{j\in[m]})$ to $\mathcal{F}_{\mathsf{corrOPE}}$ where $\vec{c}_j$ 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.* $(\mathsf{FSS.Eval}_j(i, k_i, x'))_{x'\in\{0,1\}^{|S_j|}}$.
- **Second Round:**
  3. P_0 waits to receive $(y_{i,1}, \ldots, y_{i,m})$ from $\mathcal{F}_{\mathsf{corrOPE}}$
  4. P_0 outputs $(y_{i,1}, \ldots, y_{i,m})$

[a] By [BCM22, Lemma 9], such orderings exist and furthermore can be found in expected constant time by random shuffling. Alternatively, since a random ordering of $(S_1, \ldots, S_m)$ 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 $(S_1, \ldots, S_m)$. This additional step incurs only a small additive overhead in communication, and the resulting protocol is still sublinear.

Fig. 9: Two-party protocol for obliviously evaluating shares of an additive loglog-depth FSS scheme.

Lemma 22 (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 19) 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. 9 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.

Proof. The protocol of fig. 9 essentially boils down to a single call with no interaction between the players. In the real execution, the server P_i receives no incoming communication (neither from the client P_0 nor from the ideal functionality $\mathcal{F}_{\mathsf{corrOPE}}$), therefore simulation against a corrupted server is trivial. The only message received by the client P_0 is from the ideal functionality $\mathcal{F}_{\mathsf{corrOPE}}$, but since this message is also the output of P_0, the joint view of an adversary corrupting P_0 and of the environment can be perfectly simulated by obtaining said message from the ideal functionality $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$. □

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 sections 3 and 4, this yields sublinear secure computation from compact and additive HSS. In section 5.3 we show how to extend this construction to use the weaker primitive of Las-Vegas HSS.

5.1 From Compact and Additive HSS

5.1.1 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 (*c.f.* remark 14)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. 10 a construction of loglog-depth additive FSS for $\mathcal{C}$ from single-function additive HSS for the following function coefs:

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

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 19), 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 $\{\mathsf{coefs}_{c_1,\ldots,c_N} : (c_1,\ldots,c_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$, where:

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

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 $K_1,\ldots,K_N$). Assuming the existence of compact and additive single-function HSS for any function in $\{\mathsf{coefs}_{c_1,\ldots,c_N}\}$, the construction of fig. 11 is an HSS scheme for coefs whose share generation can be distributed using low communication (with the protocol being provided in fig. 12).

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 $\vec{x} \in \{0,1\}^n$ and let $I \subseteq [n]$. If $(\vec{x}^{(1)},\ldots,\vec{x}^{(m)})$ are additive shares of $\vec{x}$, then $([\vec{x}^{(1)}]_I,\ldots,[\vec{x}^{(m)}]_I)$ are additive shares of $[\vec{x}]_I$, 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 $C = C_1\|\ldots\|C_m$ be a loglog-depth circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs and m outputs over $\mathbb{F}$ such that the following function family is $(S_1,\ldots,S_m)$-local, where $S_1,\ldots,S_m$ is some family of $(\log n/\log\log n)$-sized subsets of $[n]$:

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

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

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

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

// Note that since C has loglog-depth and $\mathcal{C}$ is $(S_1, \ldots, S_m)$-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 $(\alpha_1, \ldots, \alpha_N)$.

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

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

$\mathsf{FSS.Eval}_j(i, k_i, x')$: // $x' \in \mathbb{F}^{|S_j|}$ 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. $(p_{0,i}, p_{1,i}, \ldots) \xleftarrow{\$} \mathsf{HSS.Eval}(i, k_i)$
2. Parse $(p_{0,i}, p_{1,i}, \ldots)$ to retrieve shares $(q_{0,i}, q_{1,i}, \ldots)$ of the coefficients of $C_j(\cdot, \alpha_1, \ldots, \alpha_N)$ (*c.f.* observation 1).
3. $y_{i,j} \leftarrow (x')^{\otimes |S_j|} \cdot (q_{0,i}, q_{1,i}, \ldots)^\intercal$
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. 10: LogLog-Depth FSS Scheme from "Single-Function" Additive HSS for every LogLog-Depth Circuit.

Lemma 23 (LogLog-Depth FSS Scheme from "Single-Function" Additive HSS). *Let $N \geq 2$ be a number of parties, and let $C = C_1 \| \ldots \| C_m$ be a loglog-depth circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs and m outputs over $\mathbb{F}$ such that the following function family is $(S_1, \ldots, S_m)$-local, where $S_1, \ldots, S_m$ is some family of $(\log n / \log\log n)$-sized subsets of $[n]$:*

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

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

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

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

Then the construction of fig. 10 is an N-party additive loglog-depth and $(S_1, \ldots, S_m)$-local FSS scheme for $\mathcal{C}$.

Proof. Consider the construction $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ provided in fig. 10. The fact that $\mathsf{FSS.Eval}$ has loglog-depth and is $(S_1, \ldots, S_m)$-local follows immediately from inspection, and so does

FSS correctness (which follows from correctness of HSS). Therefore we only need to show that FSS is a secure function secret sharing scheme. Let $\mathcal{D} \subseteq [N]$. By security of HSS, for any PPT adversaries $\mathcal{A}$ and $\mathcal{A}'$,

$$\left| \Pr\left[b' = b : \begin{array}{l} (x_0, x_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda), \\ b \xleftarrow{\$} \{0,1\} \\ (x^{(1)}, \ldots, x^{(N)}) \leftarrow \mathsf{HSS.Share}(x_b) \\ b' \leftarrow \mathcal{A}'\left(\mathsf{state}, (x^{(i)})_{i \in \mathcal{D}}\right) \end{array} \right] - \frac{1}{2} \right| \leq \mathrm{negl}(\lambda).$$

In particular, for every $(\alpha_i)_{i\in[N]} \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}$, $\{(x^{(i)})_{i\in\mathcal{D}} \colon (x^{(i)})_{i\in[N]} \xleftarrow{\$} \mathsf{HSS.Share}(\alpha_1 \| \ldots \| \alpha_N)\} \overset{\mathrm{c}}{\approx} \{(x^{(i)})_{i\in\mathcal{D}} \colon (x^{(i)})_{i\in[N]} \xleftarrow{\$} \mathsf{HSS.Share}(0^{\mathbb{F}^{\ell_1}} \| \ldots \| 0^{\mathbb{F}^{\ell_N}})\}$. If we define the algorithm $\mathsf{Sim}^{\mathsf{HSS}}$ as "$\mathsf{Sim}^{\mathsf{HSS}}(1^\lambda, 1^N, 1^n, 1^m) \coloneqq ((x^{(i)})_{i\in[N]} \xleftarrow{\$} \mathsf{HSS.Share}(0^{\mathbb{F}^{\ell_1}} \| \ldots \| 0^{\mathbb{F}^{\ell_N}})$; Output $(x^{(i)})_{i\in\mathcal{D}})$" then, because the $\mathsf{FSS.Gen}$ essentially boils down to a single invocation of $\mathsf{HSS.Share}$, for every sequence of functions $(\tilde{g}_{\alpha_{1,\lambda},\ldots,\alpha_{N,\lambda}})_{\lambda\in\mathbb{N}^\star}$, $\{(k_i)_{i\in\mathcal{D}} \colon (k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, \tilde{g}_{\alpha_{1,\lambda},\ldots,\alpha_{N,\lambda}})\} \overset{\mathrm{c}}{\approx} \{\mathsf{Sim}^{\mathsf{HSS}}(1^\lambda, 1^N, 1^n, 1^m)\}$, and therefore FSS is a secure function secret sharing scheme. □

5.2.1 Securely Realising $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ in Low Communication. The FSS scheme $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ of fig. 10 is parameterised by an additive single-function HSS scheme for the function coefs. We provide in fig. 11 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 (the protocol is described in fig. 13). As explained in the overview of section 5.1.1, we use a standard hybrid encryption trick in order to build HSS for coefs from HSS for $\{\mathsf{coefs}_{c_1,\ldots,c_N}\}$. This allows us to securely distribute the shares of HSS for coefs (and hence the keys of the FSS scheme of fig. 10) by using generic secure computation to distribute the shares of HSS for $\{\mathsf{coefs}_{c_1,\ldots,c_N}\}$ (which can be done in complexity $\mathsf{poly}(\lambda + N)$).

Lemma 24 ($\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ for the LogLog-Depth FSS scheme of fig. 10 can be realised with low communication). *Let $N \geq 2$ be a number of parties, and let $C = C_1 \| \ldots \| C_m$ be a loglog-depth circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs and m outputs over $\mathbb{F}$ such that the following function family is $(S_1, \ldots, S_m)$-local, where $S_1, \ldots, S_m$ is some family of $(\log / \log\log)$-sized subsets of $[n]$:*

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

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 $\{\mathsf{coefs}_{c_1,\ldots,c_N} \colon (c_1, \ldots, c_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$, where:

$$\begin{array}{ll} \mathsf{coefs}_{c_1,\ldots,c_N} \colon \mathbb{F}^\lambda \times \cdots \times \mathbb{F}^\lambda \to \mathbb{F}^\star & \text{where } (p_0, p_1, \ldots) \text{ are the coefficients of} \\ \qquad (K_1, \ldots, K_N) \mapsto (p_0, p_1, \ldots) & \text{the polynomial representation of all the} \\ & C_j(X, c_1 - G_1(K_1), \ldots, c_N - G_N(K_N)), \\ & j \in [m] \text{ (which are polynomials in } X\text{).} \end{array}$$

Then the protocol Π_{SD} provided in fig. 13 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.

The FSS scheme $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ of fig. 10 is parameterised by an additive single-function HSS scheme for the function coefs. We provide in fig. 11 such an HSS scheme with the additional property that it yields FSS for which $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ in can be securely realised in low communication. The protocol is described in fig. 10.

5.2.2 Building HSS for coefs.

HSS Single-Function HSS for coefs

Parameters:

- For $i \in [N]$, $G_i \colon \{0,1\}^\lambda \to \mathbb{F}^{\ell_i}$ is a PRG.
- $\mathsf{HSS}' = (\mathsf{HSS}'.\mathsf{Share}, \mathsf{HSS}'.\mathsf{Eval})$ is an N-party compact and additive single-function (remark 14) HSS scheme for any function in $\{\mathsf{coefs}_{c_1,\dots,c_N} \colon (c_1,\dots,c_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$, where:

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

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

$\mathsf{HSS}.\mathsf{Share}(1^\lambda, (\alpha_1, \dots, \alpha_N))$:

1. For $i = 1, \dots, N$:
   (a) Sample a PRG seed $K_i \xleftarrow{\$} \{0,1\}^\lambda$ (for $G_i \colon \{0,1\}^\lambda \to \mathbb{F}^{\ell_i}$)
   (b) Set $c_i \leftarrow \alpha_i + G_i(K_i)$
2. $(k_1^{\mathsf{HSS}'}, \dots, k_N^{\mathsf{HSS}'}) \xleftarrow{\$} \mathsf{HSS}'.\mathsf{Share}(1^\lambda, (K_1 \| \dots \| K_N))$
3. For $i \in [N]$, set $k_i \leftarrow (K_i, k_i^{\mathsf{HSS}'}, (c_1, \dots, c_N))$
4. Output $(k_1, \dots, k_N)$

$\mathsf{HSS}.\mathsf{Eval}(i, k_i, \mathsf{coefs})$:

1. Parse k_i as $k_i = (K_i, k_i^{\mathsf{HSS}'}, (c_1, \dots, c_N))$
2. $(p_{0,i}, p_{1,i}, \dots) \xleftarrow{\$} \mathsf{HSS}.\mathsf{Eval}(i, k_i^{\mathsf{HSS}}, \mathsf{coefs}_{c_1,\dots,c_N})$
3. Output $(p_{0,i}, p_{1,i}, \dots)$

Fig. 11: HSS Scheme for coefs.

Lemma 25. *With the notations of fig. 11, if $G_1, \dots, G_N$ are PRGs and HSS' is a compact and additive single-function HSS scheme for any function in $\{\mathsf{coefs}_{c_1,\dots,c_N} \colon (c_1,\dots,c_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$, then HSS is an additive single-function HSS for $\{\mathsf{coefs}\}$.*

Proof. Let $\mathcal{D} \subsetneq [N]$, and let $\mathcal{A}, \mathcal{A}'$ be *p.p.t.* adversaries. For all $(\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}$ consider the random variables $(x_0 = (K_{1,0} \| \dots \| K_{N,0}), x_1 = (K_{1,1} \| \dots \| K_{N,1}), \mathsf{state}) \xleftarrow{\$} \mathcal{A}(1^\lambda)$, $b \xleftarrow{\$} \{0,1\}$, $(c_1, \dots, c_N) \coloneqq (\alpha_1 + G(K_1), \dots, \alpha_N + G(K_N))$, $(k_{1,b}^{\mathsf{HSS}'}, \dots, k_{N,b}^{\mathsf{HSS}'}) \xleftarrow{\$} \mathsf{HSS}'.\mathsf{Share}(x_b)$, $(x_b^{(i)})_{i \in [N]} \coloneqq (K_{i,b}, k_{i,b}^{\mathsf{HSS}'}, (c_1, \dots, c_N))_{i \in [N]}$, and $b' \xleftarrow{\$} \mathcal{A}'(\mathsf{state}, (x_b^{(i)})_{i \in \mathcal{D}})$. Observe that security of HSS is equivalent to $|\Pr[b = b'] - 1/2| \leq \mathrm{negl}(\lambda)$. By security of the PRGs $(G_i)_{i \in [N] \setminus \mathcal{D}}$, $\Pr[b = b']$ is only changed by at most an negligible additive factor if the law of $(c_i)_{i \in \mathcal{D}}$ is changed to uniformly random. Thence, the problem of showing $|\Pr[b = b'] - 1/2| \leq \mathrm{negl}(\lambda)$ in this modified experiment for HSS reduces to the security of HSS' via a standard hybrid argument. □

5.2.3 Low-Communication Protocol for $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$. We now complete the construction by providing the low-communication protocol to distribute the keys of the FSS scheme of fig. 10. The protocol itself is relatively straightforward. The parties each sample a PRG seed and use it to mask their inputs, then broadcast these masked values (using communication $\mathcal{O}(N \cdot n)$). The parties then use generic MPC to distribute HSS shares of the concatenation of the N PRG seeds (the ideal functionality is provided in fig. 12). Because the HSS scheme is compact and the input size is $N \cdot \lambda$, this step uses only $(N \cdot \lambda)^{\mathcal{O}(1)}$ bits of communication.

Functionality Distributed-Input HSS Share Distribution $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$

Parameters: The ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ is parameterised by a number of parties N and an N-party HSS scheme $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$.

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

Input: Wait to receive $(\mathtt{input}, i, x_i)$ from each party P_i.

Output: Run $(x^{(1)}, \ldots, x^{(N)}) \xleftarrow{\$} \mathsf{HSS.Share}(1^\lambda, (x_1 \| \ldots \| x_N))$; Output $x^{(i)}$ to each party P_i $(1 \leq i \leq N)$.

Fig. 12: Ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ for the generation of HSS shares of the concatenation of the parties inputs.

Protocol FSS Share Distribution Π_{SD}

Parties: $P_1, \ldots, P_N$.

Parameters:

- $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ is the N-party loglog-depth FSS scheme defined in fig. 10; $(\mathsf{FSS.Eval}_j)_{j\in[m]}$ are defined as in fig. 10. Implicitly, Π_{SD} inherits the parameters $C\colon \mathbb{F}^{\ell_0} \times \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \to \mathbb{F}^m, \mathcal{C}, (S_j)_{j\in[m]}, \mathsf{HSS}, \mathsf{coefs}$ of FSS.
- For $i \in [N]$, $G_i\colon \{0,1\}^\lambda \to \mathbb{F}^{\ell_i}$ is a constant-depth PRG.

Hybrid Model: The protocol is defined in the $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$-hybrid model.

Input: Each party P_i $(i \in [N])$ holds input $x_i \in \mathbb{F}^{\ell_i}$.

The Protocol: Each party P_i (for $i \in [N]$) does the following:

1. Sample $K_i \xleftarrow{\$} \{0,1\}^\lambda$ and set $c_i \leftarrow x_i + G_i(K_i)$
2. For $j \in [N] \setminus \{i\}$ (in parallel), send c_i to P_j and wait to receive c_j from P_j
3. Send $(\mathtt{input}, i, K_i)$ to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ and wait to receive k_i^{HSS} from $\mathcal{F}_{\mathsf{HSS\text{-}SD}}$
4. Set $k_i^{\mathsf{FSS}} \leftarrow (K_i, k_i^{\mathsf{HSS}}, (c_1, \ldots, c_N))$
5. Output k_i^{FSS}

Fig. 13: N-party protocol for the share distribution of the loglog-depth FSS scheme from additive HSS of fig. 10.

Lemma 26 ($\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ for the LogLog-Depth FSS scheme of fig. 10 can be realised with low communication). *Let $N \geq 2$ be a number of parties, and let $C = C_1 \| \ldots \| C_m$ be a loglog-depth circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs and m outputs over $\mathbb{F}$ such that the following function family is $(S_1, \ldots, S_m)$-local, where $S_1, \ldots, S_m$ is some family of $(\log / \log\log)$-sized subsets:.*

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

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:*

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

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*).*

Then the protocol Π_{SD} *provided in fig. 13 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.*

Proof. There are two separate claims in lemma 26.

- *UC-Security of* Π_{SD} *in the* $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$*-hybrid model.* Let $\mathcal{A}$ be a semi-honest, static adversary that interacts with parties $P_1, \ldots, P_N$ running protocol Π_{SD} in the $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$-hybrid model. Let us describe a simulator Sim such that no environment $\mathcal{Z}$ can distinguish with non-negligible probability whether it is interacting with $\mathcal{A}$ and Π_{SD} in the $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$-hybrid model, or with Sim and $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$. Let $\mathcal{D} \subsetneq [1, N]$ be the set of statically corrupted parties and let $(x_i)_{i \in \mathcal{D}}$ be their inputs. Sim invokes an internal copy of $\mathcal{A}$, and simulates communication with the environment in the standard way: Every input value that Sim receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment), and every output value written by $\mathcal{A}$ on its output tape is copied to Sim's own output tape (to be read by $\mathcal{Z}$). Sim simulates the protocol's execution in the following manner:
  1. For each $i \in \mathcal{D}$:
     - Sim sends $(\mathtt{input}, i, x_i)$ to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ on behalf of the corrupted party P_i, and waits to receive k_i^{FSS}.
     - Sim parses k_i^{FSS} as $k_i^{\mathsf{FSS}} = (K_i, k_i^{\mathsf{HSS}}, (c_1, \ldots, c_N))$
  2. For each $(i, j) \in ([N] \setminus \mathcal{D}) \times \mathcal{D}$, Sim writes c_i on $\mathcal{A}$'s input tape (as if the honest party P_i sent c_i to P_j).
  3. Sim simulates the ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ by writing k_i^{HSS} for $i \in \mathcal{D}$ on $\mathcal{A}$'s input tape (as if the ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ sent k_i^{HSS} to each corrupted party P_i).

  It follows from inspection that this simulation is perfect.
- *Low communication protocol in the real world.* Seminal results on completeness of OT for MPC, instantiating $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ with an oblivious transfer (protocol) can be done with communication $(N + \lambda)^{\mathcal{O}(1)}$.

□

5.3 From Compact and Additive HSS with Errors

In this section, we give a new construction of loglog-depth additive FSS scheme from additive HSS. In contrast to the construction of the previous section, this construction starts from a *Las Vegas* HSS scheme, *i.e.*, an HSS scheme where each output has a non-negligible correctness error, which can be reduced to an arbitrarily small inverse polynomial function (see Definition 15). A core ingredient of our construction, beyond a Las Vegas HSS scheme, is a *distributed point function* (DPF, [GI14]): a 2-party FSS scheme for the class of point functions $f_{\alpha,\beta} : \{0,1\}^\ell \to \mathbb{G}$ such that $f_{\alpha,\beta}(\alpha) = \beta$, and $f_{\alpha,\beta}(x) = 0$ otherwise. A sequence of works [GI14, BGI15, BGI16b] has led to highly efficient constructions of DPF schemes from any pseudorandom generator (PRG). (In fact, as one of the two parties will know the identity of the secret nonzero evaluation input, the weaker tool of a punctured pseudorandom function will suffice for the construction as given.)

Theorem 27 (PRG-based DPF [BGI16b]). *Given a length-doubling PRG* PRG*, and assuming* $|\mathbb{F}| \leq 2^{O(\ell)}$*, there exists a DPF for point functions* $f_{\alpha,\beta} : \{0,1\}^\ell \to \mathbb{F}$ *with key size* $O(\ell \cdot \lambda)$ *bits. The key generation algorithm* Gen *and the evaluation algorithm* Eval *invoke* PRG *at most* $O(\ell)$ *times.*

5.3.1 Additive FSS Scheme from Las Vegas Additive HSS. In this section, we focus our attention to the $(N+1)$-party setting with $N = 2$, in line with the fact that all known instantiations of Las Vegas HSS are for 2 parties. As in Section 5.2, we consider a circuit C with m outputs (denoting C_i the circuit computing the i-th) output and $n = \ell_0 + \ell_1 + \ell_2$ inputs, such that the functions $x \to C(\alpha_0, \alpha_1, x)$ are all $(\log / \log\log)$-local and $(\log / \log\log)$-degree (for any possible choice of (α_0, α_1); note that this holds in particular when C is itself a $(\log\log - \log\log\log)$-depth circuit). For $j = 1$ to m, we let S_j denote the $(\log n / \log\log n)$-sized subset of entries of x that influences $C(\alpha_0, \alpha_1, x)$.
Let $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ be a 2-party Las Vegas additive HSS scheme for a class $\mathcal{F}$ such that $\mathsf{coefs} \in \mathcal{F}$, where $\mathsf{coefs}(\alpha_0, \alpha_1)$ computes the coefficients of the representation of $C_j(\alpha_0, \alpha_1, \cdot)$ as multivariate polynomials for $j = 1$ to m (there are at most $\binom{2n}{n} = n(1 + o(1))$ such coefficients for each $C_j(\alpha_0, \alpha_1, \cdot)$).
Note that the definition of Las Vegas HSS (Definition 15) guarantees that the evaluation is (verifiably) correct with probability $1 - \delta$, for some inverse polynomial bound δ. Without loss of generality, when we homomorphically evaluate functions with multiple outputs, we can actually assume that *each output* fails with independent probability δ: it suffices for this to evaluate individually the function restricted to each of its output, and to use a nonce in Eval to guarantee (pseudo)-independent failure probabilities (see [BGI16a] for discussions about this). Furthermore, denoting B a bound on the total number of outputs of the target function, setting the individual failure bounds δ of each output to $1/B$ guarantees an expected constant number of failures overall. Then, by a straightforward Chernoff bound, one can assume that the total number of $\perp$ flags obtained by any party is at most λ, except with probability $\mathrm{negl}(\lambda)$. Therefore, to simplify the description, we slightly change the template definition of $\mathsf{HSS.Eval}$:

- We assume that $\mathsf{HSS.Eval}$ returns a list T of outputs, together with a list flags of all the positions of the lists for which a $\perp$ flag was raised;
- we write $\mathsf{HSS.Eval}(\mathsf{coefs}, s, \lambda)$ to indicate that coefs is homomorphically evaluated on a share s such that the *total number of* $\perp$ *flags* output by $\mathsf{HSS.Eval}$ (*i.e.* the size of flags) is bounded by λ with overwhelming probability.

LogLog-Depth FSS Scheme from Las Vegas HSS

Parameters: A 2-party Las Vegas additive HSS scheme $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ for a class $\mathcal{F}$ such that $\mathsf{coefs} \in \mathcal{F}$, where $\mathsf{coefs}(\alpha_0, \alpha_1)$ computes the (polynomially many) coefficients of the representation of $C_j(\cdot, \alpha_0, \alpha_1)$ as multivariate polynomials for $j = 1$ to m. Let B be the number of outputs of coefs. In the description below, we consider two virtual parties, P_0 and P_1. We use the expression "sample X for P_i" to indicate that X is sampled using the random tape of P_i.

$\mathsf{FSS.Gen}(1^\lambda, C(\alpha_0, \alpha_1, \cdot))$:

1. Let $(s_0, s_1) \leftarrow \mathsf{HSS.Share}(\alpha_0, \alpha_1)$. For $i \in \{0,1\}$, sample $\mathsf{seed}_i \xleftarrow{\$} \{0,1\}^\lambda$. Define $R_i \leftarrow \mathsf{PRG}(\mathsf{seed}_i)$ to be the random tape of the (virtual) party P_i.
2. For $i \in \{0,1\}$, compute $(T_i, \mathsf{flags}_i) \xleftarrow{\$} \mathsf{HSS.Eval}(\mathsf{coefs}, s_i, \lambda)$ for P_i. We assume that $|\mathsf{flags}_i| = \lambda$ (if $|\mathsf{flags}_i| > \lambda$, we restart from scratch; else, we pad flags_i with arbitrary indices until $|\mathsf{flags}_i| = \lambda$).
3. $\forall i \in \{0,1\}$, let $\mathsf{flags}_i = \{j_{i,1}, \cdots, j_{i,\lambda}\}$. For $k = 1$ to λ, set $T_i[j_{i,k}]$ to 0.
4. $\forall i \in \{0,1\}$, for $k = 1$ to λ, define $f_{i,k}$ to be the point function which evaluates to $\mathsf{coefs}_{j_{i,k}}(\alpha_0, \alpha_1) - T_{1-i}[j_{i,k}]$ on $j_{i,k}$ (and to 0 everywhere else), and run $(t^i_{0,k}, t^i_{1,k}) \xleftarrow{\$} \mathsf{DPF.Gen}(f_{i,k})$ using fresh random coins.
5. Set $\mathsf{k}_i \leftarrow (s_i, \mathsf{seed}_i, \mathsf{flags}_i, (t^0_{i,k}, t^1_{i,k})_{k \le \lambda})$ for $i = 0, 1$, and output $(\mathsf{k}_0, \mathsf{k}_1)$.

$\mathsf{FSS.Eval}_j(i, \mathsf{k}_i, x')$: we view x' as $x[S_j]$, *i.e.*, the size-$(\log n / \log\log n)$ subset of entries of some vector x, indexed by S_j. Parse k_i as $(s_i, \mathsf{seed}_i, \mathsf{flags}_i, (t^i_{i,k}, t^i_{1-i,k})_{k \le \lambda})$. In the following, use $R_i = \mathsf{PRG}(\mathsf{seed}_i)$ as random tape, matching the coins used in $\mathsf{FSS.Gen}$.

1. Compute $(T_i, \mathsf{flags}_i) \xleftarrow{\$} \mathsf{HSS.Eval}(\mathsf{coefs}, s_i, \lambda)$. For each $j \in \mathsf{flags}_i$, set $T_i[j]$ to 0.

2. For $k = 1$ to λ, for $j = 1$ to B, compute $c^j_{i,k} \leftarrow \mathsf{DPF.Gen}(t^0_{i,k}, j) + \mathsf{DPF.Gen}(t^1_{i,k}, j)$. Set $T_i[j] \leftarrow T_i[j] + \sum_{k=1}^{\lambda} c^j_{i,k}$.
3. Parse T_i to retrieve shares $(q_{0,i}, q_{1,i}, \dots)$ of the coefficients of $C_j(\alpha_0, \alpha_1, \cdot)$ (such parsing is possible because the shares are additive, *c.f.* observation 1).
4. Set $y_{i,j} \leftarrow (x')^{\otimes |S_j|} \cdot (q_{0,i}, q_{1,i}, \dots)^\intercal$.
5. Output $y_{i,j}$

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

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

Fig. 14: LogLog-Depth FSS Scheme from Las Vegas Additive HSS for every LogLog-Depth Circuit.

Lemma 28 (LogLog-Depth FSS Scheme from Las Vegas HSS). *Let $C = C_1 \| \dots \| C_m$ be a circuit with $n = \ell_0 + \ell_1 + \ell_2$ satisfying the loglog-depth constraint described above. Let $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ be a 2-party Las Vegas additive HSS scheme for a function class containing the coefs function. Then the construction of fig. 14 is a 2-party additive loglog-depth FSS scheme for $\mathcal{C}$.*

Proof. Consider the construction $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ provided in fig. 14. loglog-depth is identical to that of the previous construction (from Figure fig. 10). Correctness follows from the correctness of the HSS scheme, and from that of the DPF scheme. Concretely:

1. With probability 1, after computing $(T_0, \mathsf{flags}_0) \xleftarrow{\$} \mathsf{HSS.Eval}(\mathsf{coefs}, s_0, \lambda)$ and $(T_1, \mathsf{flags}_1) \xleftarrow{\$} \mathsf{HSS.Eval}(\mathsf{coefs}, s_1, \lambda)$, for every $j \notin \mathsf{flags}_0 \cup \mathsf{flags}_1$, $T_0[j] + T_1[j] = \mathsf{coefs}_j(\alpha_0, \alpha_1)$.
2. After updating $T_i[j]$ to $T_i[j] + \sum_{k=1}^{\lambda} c_{i,k}$ (in step 2), we have

$$T_0[j] + T_1[j] = \mathsf{coefs}_j(\alpha_0, \alpha_1) + \sum_{k=1}^{\lambda} (c^j_{0,k} + c^j_{1,k}),$$

   and it holds that $c^j_{0,k} + c^j_{1,k} = f_{0,k}(j) + f_{1,k}(j) = 0$, by correctness of the DPF scheme, and since $f_{0,k}$ and $f_{1,k}$ are point functions which (in particular) evaluate to 0 on all $j \notin \mathsf{flags}_0 \cup \mathsf{flags}_1$.
3. For every $j \in \mathsf{flags}_0 \setminus \mathsf{flags}_1$, we have $T_0[j] = 0$ after step 1, and $T_1[j]$ equal to some value v. Then after step 2, we have

$$T_0[j] + T_1[j] = v + \sum_{k=1}^{\lambda} (c^j_{0,k} + c^j_{1,k}),$$

   where $c^j_{0,k} + c^j_{1,k} = f_{0,k}(j) + f_{1,k}(j)$. Now, since $j \notin \mathsf{flags}_1$, $f_{1,k}(j) = 0$ for all k, and since $j \in \mathsf{flags}_0$, there is exactly one k such that $f_{0,k}(j) \neq 0$. For this k, $f_{0,k} = \mathsf{coefs}_j(\alpha_0, \alpha_1) - v$. Therefore, $T_0[j] + T_1[j] = v + \mathsf{coefs}_j(\alpha_0, \alpha_1) - v$.
4. The cases $j \in \mathsf{flags}_1 \setminus \mathsf{flags}_0$ and $j \in \mathsf{flags}_0 \cap \mathsf{flags}_1$ are similar (in the latter, there will be exactly one k where $f_{0,k}(j)$ is non-zero, and one k', possibly equal to k, where $f_{0,k'}(j)$ is non-zero).

Therefore, the $(T_0[j], T_1[j])_j$ are indeed additive shares of the outputs of coefs on input (α_0, α_1), hence $(x')^{\otimes |S_j|} \cdot (q_{0,i}, q_{1,i}, \cdots)^\intercal = Q(x')$, where Q is the $|S_j|$-variate polynomials computing $C_j(\alpha_0, \alpha_1, x)$ for any x such that $x' = x[S_j]$.
We clarify a minor technicality regarding the analysis above: since Las Vegas HSS has probability 1 of yielding correct shares of the output whenever the flags are equal to $\top$, this remains true when the randomness of Eval is sampled as $R_i \leftarrow \mathsf{PRG}(\mathsf{seed}_i)$ for party P_i. However, to guarantee correctness, we must also show that with overwhelming probability, $|\mathsf{flags}_i| \leq \lambda$. When using true random coins, this holds with overwhelming probability by a straightforward Chernoff bound. When using pseudorandom coins, as we do here, this *statistical* property holds under the assumption that the PRG is secure (it is a standard fact that the output of a secure PRG must pass all polynomial-time verifiable statistical

tests); hence, correctness holds under the existence of a secure PRG. This concludes the proof of correctness.
It remains to show that FSS is a secure function secret sharing scheme. As before, it follows from the fact that HSS is a secure homomorphic secret sharing scheme for which we have a simulator $\mathsf{Sim}^{\mathsf{HSS}}$. We additionally mask each output with 2λ outputs of (2λ independent instances of) a DPF, which can also be simulated using the simulator $\mathsf{Sim}^{\mathsf{DPF}}$. □

5.3.2 Securely Realising Gen in Low Communication. We describe a low-communication two-party protocol for securely distributing $(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, C(\alpha_0, \alpha_1, \cdot))$ between two parties, P_0 and P_1, holding α_0 and α_1 as respective inputs, in the honest-but-curious model.

Private information retrieval. A private information retrieval is a two party protocol between a server S holding a vector z (the database) and a client C holding an integer i, where only the client receives an output. The security parameter λ and the length $n(\lambda) = \mathsf{poly}(\lambda) = |z|$ of the server database are a common (public) input. We let $\mathsf{View}_S(\lambda, z, i)$ denote the view of S during its interaction with C on respective inputs (z, i) with common input $(\lambda, n = |z|)$, and by $\mathsf{Out}_C(\lambda, z, i)$ the output of C after the interaction.

Definition 29 (Private Information Retrieval). *A private information retrieval for database size $n = n(\lambda)$ (n-PIR) is an interactive protocol between a PPT server S holding a vector $z \in \mathbb{F}^n$ and a PPT client C holding an index $i \leq n$ which satisfies the following properties:*

- ***Correctness:*** *there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, $z \in \{0,1\}^n$, $i \in [n]$:*
$$\Pr[\mathsf{Out}_C(\lambda, z, i) = z_i] \geq 1 - \mu(\lambda).$$
- ***Security:*** *there exists a negligible function μ such that for every PPT adversary $\mathcal{A}$, large enough $\lambda \in \mathbb{N}$, $(i, j) \in [n]^2$, and $z \in \{0,1\}^n$:*
$$|\Pr[\mathcal{A}(1^{\lambda+n}, \mathsf{View}_S(\lambda, z, i)) = 1] - \Pr[\mathcal{A}(1^{\lambda+n}, \mathsf{View}_S(\lambda, z, j)) = 1]| \leq \mu(\lambda, n).$$
- ***Efficiency:*** *A PIR is* polylogarithmic *if its communication complexity $c(\lambda, n)$, measured as the worst-case number of bits exchanged between S and C (over their inputs (z, i) and their random coins), satisfies $c(\lambda, n) = \mathsf{poly}(\lambda, \log n)$.*

We define a private *shared* information retrieval (PSIR) analogeously to a private information retrieval, except that the client C with input i and the server S with input z get as output random shares of z_i.

The protocol. As in the previous constructions, we assume that there is a succinct protocol (with communication linear in the input size, up to an additive $\mathsf{poly}(\lambda)$ term) for distributing the shares of HSS. Such succinct protocols are known for all known Las Vegas HSS. Furthermore, we assume that $\mathsf{HSS.Eval}$ can be run on tuples of input shares (generated separately), rather than individual input shares; that is, given $(x^{(0)}, x^{(1)}) \leftarrow \mathsf{HSS.Share}(1^\lambda, x)$ and $(y^{(0)}, y^{(1)}) \leftarrow \mathsf{HSS.Share}(1^\lambda, y)$, $\mathsf{Eval}(i, f, (x^{(i)}, y^{(i)}))$ for $i = 0, 1$ produces shares of $f(x, y)$. We note that known constructions of Las Vegas HSS [BGI16a, BKS19] have a setup phase that generates public parameters, such that $\mathsf{HSS.Eval}$ can be run on any tuple of HSS shares generated from the same public parameters.
The protocol operates in a hybrid model, with access to the following functionalities:

- A share distribution functionality $\mathcal{F}_{\mathsf{HSS\text{-}SD}}$ (as given on Figure fig. 12).
- A private shared information retrieval functionality $\mathcal{F}_{\mathsf{PSIR}}$ which receives i from a client, and $(z, r) \in \mathbb{F}^n \times \mathbb{F}$ from a server. It outputs $z_i - r$ to the client, and nothing to the server. (Note that given a PIR, our reduction from PIR to PSIR securely instantiates this functionality).
- An ideal 'selection' functionality $\mathcal{F}_{\mathsf{sel}}$ which, given $(v_1^b, \cdots, v_\lambda^b) \in \mathbb{F}^\lambda$ and $(\mathsf{flag}_1^b, \cdots, \mathsf{flag}_\lambda^b) \in \{\top, \bot\}^\lambda$ from each party P_b, returns uniformly random shares of $v_i^0 + v_i^1$, where i is the first index such that $\mathsf{flag}_i^0 = \mathsf{flag}_i^1 = \top$.
- An ideal functionality $\mathcal{F}_{\mathsf{DPF}}$ which distributes DPF keys given shares of (the description of) a point function:
  - $\mathcal{F}_{\mathsf{DPF}}$ takes as input (w^b, j) from one party P_b, and $(w^{1-b}, \bot)$ from P_{1-b};
  - it defines f the point function which evaluates to $w = w^0 + w^1$ on input j (and to 0 elsewhere);
  - it outputs $(t^0, t^1) \xleftarrow{\$} \mathsf{DPF.Gen}(f)$ to P_0 and P_1.

Succinct Protocol for the LogLog-Depth FSS Scheme from Las Vegas HSS

1. P_0 and P_1 call $\mathcal{F}_{\mathsf{HSS\text{-}SD}}$ on respective inputs α_0, α_1, and receive $(s_0, s_1) \xleftarrow{\$} \mathsf{HSS.Share}(\alpha_0, \alpha_1)$. Each party P_i picks $\mathsf{seed}_i \xleftarrow{\$} \{0,1\}^\lambda$ and stretches $R_i \leftarrow \mathsf{PRG}(\mathsf{seed}_i)$.
2. For $i \in \{0,1\}$, P_i samples $(T_i, \mathsf{flags}_i) \xleftarrow{\$} \mathsf{HSS.Eval}(\mathsf{coefs}, s_i, \lambda)$ using R_i as random tape. We assume that $|\mathsf{flags}_i| = \lambda$ (if $|\mathsf{flag}_i| > \lambda$, P_i aborts the protocol; else, P_i pads $|\mathsf{flag}_i|$ with dummy items until $|\mathsf{flags}_i| = \lambda$).
3. For $i = 0, 1$, and for $k = 1$ to λ, denoting $\{j_{i,1}, \cdots, j_{i,\lambda}\} = \mathsf{flags}_i$, P_i sets $T_i[j_{i,k}]$ to 0.
4. For $i = 0, 1$, and for $k = 1$ to λ,
   (a) P_i and P_{1-i} call $\mathcal{F}_{\mathsf{PSIR}}$, where P_i plays the role of the client with input $j_{i,k}$, and P_{1-i} samples $u_{i,k}^{1-i} \xleftarrow{\$} \mathbb{F}$ and plays the role of the server with inputs $(T_{1-i}, u_{i,k}^{1-i})$. Let $u_{i,k}^i$ denote P_i's output. Note that by correctness of the PSIR scheme, $u_{i,k}^0 + u_{i,k}^1 = T_i[j_{i,k}]$.
   (b) P_i and P_{1-i} call $\mathcal{F}_{\mathsf{HSS\text{-}SD}}$ on input $j_{i,k}$ from P_i, and get respective outputs $(s_{i,k}^0, s_{i,k}^1) \xleftarrow{\$} \mathsf{HSS.Share}(j_{i,k})$.
   (c) Let $\mathsf{coef}(\alpha_0, \alpha_1, j)$ denote the function computing $\mathsf{coefs}_j(\alpha_0, \alpha_1)$. Each party P_b locally run λ independent instances $(v_{i,k,\ell}^b, \mathsf{flag}_{i,k,\ell}^b) \xleftarrow{\$} \mathsf{HSS.Eval}(\mathsf{coef}, (s_b, s_{i,k}^b), 1/2)$ for $b = 0, 1$ and $\ell = 1$ to λ.
   (d) P_0 and P_1 call $\mathcal{F}_{\mathsf{sel}}$, where P_b's inputs are all candidate shares $(v_{i,k,\ell}^b)_{\ell \le \lambda}$ and corresponding flags $(\mathsf{flag}_{i,k,\ell}^b)_{\ell \le \lambda}$. Recall that $\mathcal{F}_{\mathsf{sel}}$ identifies the first ℓ such that $\mathsf{flag}_{i,k,\ell}^0 = \mathsf{flag}_{i,k,\ell}^1 = \top$ (which exists with overwhelming probability), reconstructs $v_{i,k} = v_{i,k,\ell}^0 + v_{i,k,\ell}^1$ (which is equal to $\mathsf{coef}(\alpha_0, \alpha_1, j_{i,k}) = \mathsf{coefs}_{j_{i,k}}(\alpha_0, \alpha_1)$, by correctness of the HSS scheme), and outputs fresh random shares $(v_{i,k}^0, v_{i,k}^1)$ of $v_{i,k}$ to P_0 and P_1.
   (e) P_0 and P_1 call $\mathcal{F}_{\mathsf{DPF}}$, where each party P_b has an $w_{i,k}^b \leftarrow v_{i,k}^b - u_{i,k}^b$; P_i has additional input $j_{i,k}$, and P_{1-i} has additional input $\perp$. $\mathcal{F}_{\mathsf{DPF}}$ outputs $(t_{i,k}^0, t_{i,k}^1) \xleftarrow{\$} \mathsf{DPF.Gen}(f_{i,k})$ to P_0 and P_1, which are DPF shares of $f_{i,k}$, the point function evaluating to $w_{i,k} = w_{i,k}^0 + w_{i,k}^1$ on $j_{i,k}$, and 0 elsewhere.
   (f) Each party P_b outputs $\mathsf{k}_b \leftarrow (s_b, \mathsf{seed}_b, \mathsf{flags}_b, (t_{b,k}^0, t_{b,k}^1)_{k \le \lambda})$.

Fig. 15: A succinct protocol for distributing shares of the loglog-depth FSS scheme of Figure fig. 14

Theorem 30. *Assume that* $(\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *is a secure Las Vegas additive HSS scheme where* $\mathsf{HSS.Eval}$ *can run on tuples of HSS shares. Then the protocol from Figure fig. 15 securely computes the procedure* $\mathsf{FSS.Gen}(1^\lambda, C(\alpha_0, \alpha_1, \cdot))$ *of Figure fig. 14 in the* $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}, \mathcal{F}_{\mathsf{PSIR}}, \mathcal{F}_{\mathsf{sel}}, \mathcal{F}_{\mathsf{DPF}})$*-hybrid model. Furthermore, instantiating* $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ *with a succinct HSS share distribution protocol (with communication* $\ell_0 + \ell_1 + \mathsf{poly}(\lambda)$*),* $\mathcal{F}_{\mathsf{PSIR}}$ *with a PSIR with polylogarithmic efficiency, and* $\mathcal{F}_{\mathsf{sel}}, \mathcal{F}_{\mathsf{DPF}}$ *with generic secure computation protocols, yield a protocol with total communication* $n + \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(n)$.

Proof. Correctness follows by inspection: steps (1) to (3) directly emulate the step (1) to (3) of $\mathsf{FSS.Gen}$. In step (4.a), by correctness of the PSIR, the parties obtain additive shares $(u_{i,k}^0, u_{i,k}^1)$ of $T_i[j_{i,k}]$. Step (4.b) to (4.d) let the parties generate additive shares of the value $\mathsf{coefs}_{j_{i,k}}(\alpha_0, \alpha_1)$ – that is, the value that should have been shared in $(T_0[j_{i,k}], T_1[j_{i,k}])$ if there was no error. This is done by first secure sharing $j_{i,k}$ with $\mathsf{HSS.Share}$, then running λ times the $\mathsf{HSS.Eval}$ algorithm with error probability $1/2$. Except with probability $1/2^\lambda$, it necessarily holds that on one of the λ executions, both parties got a $\top$ flag (indicating no error). Step (4.d) runs a generic protocol which, given the λ outputs of each party and their flags, identifies such an execution, reconstructs the output (which is equal to $\mathsf{coefs}_{j_{i,k}}(\alpha_0, \alpha_1)$ by Las Vegas correctness of HSS), and re-shares it (this is necessary to avoid leaking the position of the first correct share). The generic protocol in (4.e) outputs keys for a point function $f_{i,k}$ which evaluates to $w_{i,k}$ on $j_{i,k}$. We have

$$w_{i,k} = w_{i,k}^0 + w_{i,k}^1 = -u_{i,k}^0 - u_{i,k}^1 + v_{i,k}^0 + v_{i,k}^1 = -T_i[j_{i,k}] + \mathsf{coefs}_{j_{i,k}}(\alpha_0, \alpha_1),$$

hence $f_{i,k}$ matches its definition in $\mathsf{FSS.Gen}$. This concludes the proof of correctness. We turn our attention to security. Assume that P_1 is corrupted; the other case follows symmetrically. The simulator Sim, given the target output $\mathsf{k}_1 \leftarrow (s_1, \mathsf{seed}_1, \mathsf{flags}_1, (t_{1,k}^0, t_{1,k}^1)_{k \le \lambda})$ of P_1, emulates P_0 as follows (note that steps 2, 3, and 4.c only require local computation):

– (Step 1) It emulates $\mathcal{F}_{\mathsf{HSS\text{-}SD}}$ and sets P_1's output to s_1.
– (Step 4.a) It emulates the λ invocations of $\mathcal{F}_{\mathsf{PSIR}}$ where P_1 plays the role of the client (*i.e.* when $i = 1$), sampling P_1's output $u^1_{1,k}$ uniformly over $\mathbb{F}$.
– (Step 4.b) For $i = 0, 1$ and $k = 1$ to λ, it uses $\mathsf{Sim}^{\mathsf{HSS}}$ to simulate P_1's HSS share of $j_{i,k}$. Then, it emulates $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{SD}}$ and sets P_1's outputs to be the simulated shares of $j_{i,k}$.
– (Step 4.d) It emulates the 2λ invocations of $\mathcal{F}_{\mathsf{sel}}$, and set P_1's outputs from the functionality to independent uniformly random values.
– (Step 4.e) It emulates the 2λ invocations of $\mathcal{F}_{\mathsf{DPF}}$, and set P_1's outputs from the functionality to $(t^0_{1,k}, t^1_{1,k})_{k\leq\lambda}$.

Security follows from a sequence of straightforward hybrids: H_0 is the initial game. $H_1^{i,k}$ replaced the HSS share of $j_{i,k}$ by a simulated HSS share using $\mathsf{Sim}^{\mathsf{HSS}}$, for $i = 0$ to 1 and $k = 1$ to λ. Indistinguishability follows by 2λ invocations of the HSS security. H_2 replaces the λ invocations of $\mathcal{F}_{\mathsf{PSIR}}$ where P_1 plays the role of the client by emulations of the functionality with a uniformly random output $u^1_{1,k}$; this game is perfectly indistinguishable from $H_1^{1,\lambda}$, since the outputs are distributed exactly as in the honest game. H_3 replaces the 2λ invocations of $\mathcal{F}_{\mathsf{sel}}$ by emulations of the functionality with uniformly random outputs, which is perfectly indistinguishable from H_2, since the outputs of $\mathcal{F}_{\mathsf{sel}}$ are distributed exactly as in H_2. Eventually, H_3 replaces the 2λ invocations of $\mathcal{F}_{\mathsf{DPF}}$ by emulations of the functionality with outputs $t^i_{1,k}$ for $i = 0, 1$ and $k = 1$ to λ. Since the outputs are the same as in H_3, this game is perfectly indistinguishable from the previous game. This concludes the proof. □

6 Instantiations

In section 6.1, we combine the results of sections 3 to 5 and achieve sublinear secure computation from generic assumptions (HSS and forms of PIR/OLE). In section 6.2, we build four-party compact and additive HSS for loglog-depth correlations from standard assumptions (DCR and constant-locality PRGs). In section 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 $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{OE}}$ for local FSS schemes can be based on batch OPE (with correlated inputs) and section 5 builds local FSS schemes (such that $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}$ can be realised with low communication) from additive HSS (with or without errors). Plugging these two constructions into the template of section 3 yields sublinear secure multiparty computation from batch OPE and additive HSS.

Theorem 31 (Sublinear-Communication Secure $(N+1)$-Party Computation of Shallow Circuits). *Let $N \geq 2$ be a number of parties, and let $C\colon \mathbb{F}^n \to \mathbb{F}^m$ 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 $G_i\colon \{0,1\}^\lambda \to \mathbb{F}^{\ell_i}$ for $i \in [N]$,*
– *An N-party compact and additive single-function HSS scheme for any function in the class $\{\mathsf{coefs}_{\alpha_1,\ldots,\alpha_N}\colon (\alpha_1,\ldots,\alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$, 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 31 is obtained by combining the results of sections 3 to 5. Our starting point is the generic template of theorem 18 in the $(\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}, \mathcal{F}^{\mathsf{FSS}}_{\mathsf{OE}})$-hybrid model, which uses $N \cdot m \cdot \log|\mathbb{F}|$ bits of communication and makes a single call to $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}$, and N to $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{OE}}$. We use the FSS scheme of lemmas 23 and 26, for which, by lemma 22, each call to $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{OE}}$ can be implemented using communication $\mathsf{Comm}_{\mathsf{corrOPE}}(m, 2^d, 2^d, n)$ and the single call to $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}$ 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 [COS+22]. 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 $G_{\mathsf{in}}(f) : x^{(b,c)} \to \mathsf{HSS}_{\mathsf{in}}.\mathsf{Eval}(b, f, x^{(b,c)})$ 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 state the resulting theorem in theorem 32.

Theorem 32 (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 $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ of fig. 12 for the generation of HSS shares of the concatenation of the parties inputs.*

6.2.1 4-party HSS from DCR. The work of [COS+22] shows how to instantiate the above template, using the recent DCR-based HSS scheme of [OSY21] to instantiate both $\mathsf{HSS}_{\mathsf{in}}$ and $\mathsf{HSS}_{\mathsf{out}}$, when the class $\mathcal{F}$ is restricted to constant-degree multivariate polynomials. Consider for simplicity the evaluation of a single degree-c monomial $\prod_{i=1}^{c} x_i$ (handling arbitrary polynomials is done by computing shares of the deg-c monomials separately, and summing the shares). Fix a modulus N for the Paillier encryption scheme, and let d be the Paillier secret key, *i.e.*, an integer such that given any Paillier encryption $C = (1 + N)^x R^n \bmod N^2$ of an input x, we have $C^d = (1 + N)^m \bmod N^2$.
In the scheme of [OSY21], an HSS share of x_i contains (1) additive shares (over $\mathbb{Z}$) of the secret key d and (2) Paillier encryption $C_i, (C_{i,j})_{j \le |d|}$ of $x_i, (x_i \cdot d_j)_{j \le |d|}$, where the d_j are the bits of d. A core observation is that the ciphertexts $C_i, C_{i,j}$ can remain public (in the sense that they will be directly included in all 4-party shares, and not reshared), since they are included in all shares: only the shares of d must be re-shared (bitwise) with the outer scheme. Now, the homomorphic evaluation of $f(x_1, \cdots, x_c) = \prod_{i=1}^{c} x_i$ consists in $c - 1$ sequential homomorphic multiplication, where each homomorphic multiplication boils down to parallel calls to the function

$$\mathsf{Mul}_{C,v,N} : d \to \mathsf{DDLog}(C^d \bmod N^2) + v \bmod N,$$

where v is a value known to both partie (concretely, $v = F_k(\mathsf{id})$ where id is public and k is a PRF key known to both parties, which can be included "in the clear" in the four 4-party shares), DDLog is the function $\mathsf{DDLog} : X \to \lfloor X/N \rfloor \cdot (X \bmod N)^{-1} \bmod N$, and C is one of the Paillier ciphertexts. Now, since the ciphertexts C are known during the homomorphic evaluation of $\mathsf{Mul}_{C,v,N}$ by the outer scheme, we can rewrite

$$C^d = \prod_{j=1}^{|d|} (C^{2^{j-1}})^{d_j} \bmod N^2, \qquad (C^d \bmod N)^{-1} = \prod_{j=1}^{|d|} (C^{-2^{j-1}})^{d_j} \bmod N^2.$$

Since iterated products, modular reductions, rounding, and integer division are all in NC^1, the entire computation of Mul is therefore an NC^1 function (and so are parallel calls to Mul). From here, [COS+22] concludes that for every constant c, the homomorphic computation of $\prod_{i=1}^{c} x_i$ inside the inner HSS remains an NC^1 function overall. Therefore, it suffices for the outer HSS to use another DCR-based HSS (with a different Paillier modulus $N' = O(N)$), since the latter handles all NC^1 computations.

6.2.2 Handling loglog-depth circuits. The above nesting approach is limited to constant degree polynomials. At a high level, this stems from the fact that computing c sequential homomorphic Mul requires a runtime of the form $\mathsf{poly}(\lambda)^c$, where $\mathsf{poly}(\lambda)$ denotes the runtime of computing a single Mul operation. This overhead stems from the fact that the outer HSS natively handles only functions represented as restricted multiplication straight-line (RMS) programs, and converting an arbitrary circuit to an RMS program incurs a cost which is exponential in the circuit depth. The inputs to (the parallel calls to) Mul have size $\mathsf{poly}(\log N) = \mathsf{poly}(\lambda)$, the depth of the Mul circuit (which is in NC^1) is therefore $O(\log \lambda)$, hence the overall depth of the homomorphic computation is $O(c \cdot \log \lambda)$, hence the $\mathsf{poly}(\lambda)^c = 2^{O(c \cdot \log \lambda)}$ overhead.
To circumvent this limitation, we rely on complexity leveraging, and make $\log N$ slightly subpolynomial in λ. Looking ahead, this will translate to assuming the superpolynomial hardness of the DCR assumption. Writing $\log N = \kappa$ and by the above analysis, the overall runtime of evaluating a degree-c monomial is $\kappa^{\mathsf{cst} \cdot c}$, for an appropriate constant cst. Setting $\kappa \leftarrow \lambda^{1/c}$, the above becomes polynomial in λ, at the cost of assuming that DCR is secure against $\mathsf{poly}(\lambda) = \kappa^{O(c)}$ adversaries. For example, setting $c = \log(\lambda) = O(\log \kappa)$ means that we need the DCR assumption to hold against adversaries running in time $\kappa^{O(\log \kappa)}$, *i.e.*, assuming the superpolynomial hardness of DCR. Now, over $\mathbb{F}_2$, any c-local function (hence, for example, every $\log c$-depth boolean circuit) can be written as a sum of monomials of degree at most c.

6.2.3 Compactness and succinct share distribution. The above scheme is not compact: a 4-party HSS share of a length-m vector $\vec{x}$ has size $\mathsf{poly}(\kappa) \cdot m$. We show here how to transform it into a compact scheme, with share size $m + o(m) \cdot \mathsf{poly}(\kappa)$, assuming the existence of local pseudorandom generators. Local PRGs have been introduced in a seminal work of Goldreich [Gol00]. Since their introduction, they have been investigated in numerous works [MST03, AHI04, CEMT09, CEMT14, OW14, AL16, CDM+18], and are regularly used to achieve advanced cryptographic primitives, such as secure computation with constant computational overhead [IKOS08, ADI+17] or more recently indistinguishability obfuscation [JLS21, JLS22]. A local PRG G with stretch s stretches a seed seed of size t into a length-t^{1+s} pseudorandom string, such that every bit of $G(\mathsf{seed})$ is a function of a constant number of bits of seed.
To achieve compactness, we use a standard *hybrid encapsulation* technique with a local PRG. Concretely, let HSS be the non-compact 4-party HSS scheme constructed above. We construct a new scheme HSS' as follows:

- $\mathsf{HSS}'.\mathsf{Share}(x)$: given a length-m vector x of inputs, sample four seeds $(\mathsf{seed}_i)_{i \leq 4}$ for a local PRG G with stretch s and seed size $m^{1/(1+s)}$. Compute $(x^{(i)})_{i \leq 4} \leftarrow \mathsf{HSS}.\mathsf{Share}(\mathsf{seed}_1 || \mathsf{seed}_2 || \mathsf{seed}_3 || \mathsf{seed}_4)$, and $y \leftarrow x \oplus G(\mathsf{seed}_1) \oplus G(\mathsf{seed}_2) \oplus G(\mathsf{seed}_3) \oplus G(\mathsf{seed}_4)$. Output $(x^{(i)}, z)$ to each party i.
- $\mathsf{HSS}'.\mathsf{Eval}(i, f, (x^{(i)}, z))$: define the function

$$f'_z : (\mathsf{seed}_i)_{i \leq 4} \to f\left(z \oplus \bigoplus_{i=1}^{4} G(\mathsf{seed}_i)\right).$$

  Output $y^{(i)} \leftarrow \mathsf{HSS}.\mathsf{Eval}(i, f'_z, x^{(i)})$.

Correctness is clear by inspection. The security analysis of HSS' proceeds in two simple games: in the first game, using the security of HSS, simulate the shares of $(\mathsf{seed}_1, \cdots, \mathsf{seed}_4)$. In the second game, since the seeds seed_i are not used anymore, invoke the pseudorandomness of G to replace z by a uniformly random string. For efficiency, observe that since G has constant locality (and constant depth), whenever f is a loglog-depth function, so is f'.
It remain to discuss compactness. The size of a share is $|x^{(i)}| + |z| = \mathsf{poly}(\kappa) \cdot m^{1/(1+s)} + m = m + o(m) \cdot \mathsf{poly}(\kappa)$. Furthermore, the scheme admits a straightforward succinct protocol for securely distributing shares of an input vector x. Assume that the parties P_i have shares x_i of the input vector x (the case where x is the concatenation of their joint input is directly implied by this case). Each party P_i locally samples seed_i, and the party jointly run a generic secure computation protocol (e.g. using DCR-based oblivious transfer, with security parameter κ) for securely computing $\mathsf{HSS}.\mathsf{Share}(\mathsf{seed}_1 || \mathsf{seed}_2 || \mathsf{seed}_3 || \mathsf{seed}_4)$. Then, each party P_i broadcasts $z_i \leftarrow x_i \oplus G(\mathsf{seed}_i)$, and all parties reconstruct $z = \bigoplus_{i=1}^{4} z_i$. The total communication of the protocol is $4m + \mathsf{poly}(\kappa, o(m))$, which is $m \cdot (4 + o(1))$ for a large enough m.

6.3 Sublinear-Communication Secure Multiparty Computation from New Assumptions

Combining section 6.1 with instantiations of $\mathsf{corrSPIR}$ and additive HSS from the literature (and section 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 [BGI16a, Cou19, CM21] 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 33 (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 34.

- **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 super-polynomial 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 34.*

- **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 [BBDP22, OSY21] seems to implicitly yield rate-1 batch OT from DCR, and in turn correlated SPIR [BCM22]: 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.

Proof.

- **3-PC:** The first, DDH-based, instantiation follows the template of building loglog-depth FSS from additive Las-Vegas HSS, and all others follow the template of building loglog-depth FSS from additive HSS directly.
  - *From DDH and LPN:* We instantiate the template of section 3 (parameterised by an FSS scheme, in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid model) with the loglog-depth FSS scheme of lemma 28, which we in turn instantiate using the Las-Vegas HSS scheme of [BGI16a]. By theorem 30, $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ can be realised in communication $n + \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(n)$ using polylog PSIR. Recall from section section 5.3 that any PIR scheme yields a PSIR scheme with the same communication complexity using a straightforward reduction: S picks $r \xleftarrow{\$} \mathbb{F}$ and substracts r to all entries of z, getting a new vector z'. Then, C and S use a PIR scheme with respective inputs i and z'. C outputs z'_i and S outputs r, which form random shares of z_i, as desired. We can therefore use the polylogarithmic PIR protocol from DDH in [DGI+19]. Finally, we use the two-round correlated SPIR with mix-and-match queries in [BCM22] from DDH (and based on the rate-1 batch-OT from DDH in [BBDP22]).
  - *From polynomial-modulus LWE and LPN:* We instantiate theorem 31 using the additive HSS scheme for NC^1 of [BKS19] from polynomial-modulus LWE and the two-round correlated SPIR with mix-and-match queries in [BCM22] from polynomial-modulus LWE and LPN (and based on the rate-1 batch-OT from polynomial-modulus LWE and LPN in [BBDP22]).
  - *From QR and Superpolynomial LPN:* We instantiate theorem 31 using the single-function additive HSS scheme for any $(\log\log s)/4$-depth circuit from from quasi-polynomial LPN implicitly present in [CM21] (and based on the PCG for "subsets tensor powers") and the two-round correlated SPIR QR-modulus LWE and LPN (and based on the rate-1 batch-OT from polynomial-modulus LWE and LPN in [BBDP22]).
- **5-PC:** We instantiate theorem 31 using the four-party additive HSS scheme for NC^1 of section 6.2.1 from DCR and the two-round correlated SPIR with mix-and-match queries in [BCM22] from LPN and any of polynomial-modulus LWE, DDH, or QR.

□

We conclude by remarking that while this may not be immediately apparent due to the complicated expressions, the communication complexities from theorem 33 do indeed qualify as "sublinear in the circuit-size".

Remark 34 (The Expressions of Theorem 33 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 $\lambda^{O(1)} + \mathcal{O}(n+m+s^{1+2\delta})$. 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 proof-reading, 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

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

ADOS22. Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. *Cryptology ePrint Archive*, 2022.

AHI04. Michael Alekhnovich, Edward A. Hirsch, and Dmitry Itsykson. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *ICALP 2004*, volume 3142 of *LNCS*, pages 84–96. Springer, Heidelberg, July 2004.

AIK09. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. *Journal of Cryptology*, 22(4):429–469, October 2009.

AJL+12. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Heidelberg, April 2012.

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

BBDP22. Zvika Brakerski, Pedro Branco, Nico Döttling, and Sihang Pu. Batch-OT with optimal rate. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 157–186. Springer, Heidelberg, May / June 2022.

BCM22. Elette Boyle, Geoffroy Couteau, and Pierre Meyer. Sublinear secure computation from new assumptions. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 121–150. Springer, Heidelberg, November 2022.

Bel84. Edward G. Belaga. Locally synchronous complexity in the light of the trans-box method. In M. Fontet and K. Mehlhorn, editors, *STACS 84*, pages 129–139, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.

BFKL94. Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 278–291. Springer, Heidelberg, August 1994.

BFKR91. Donald Beaver, Joan Feigenbaum, Joe Kilian, and Phillip Rogaway. Security with low communication overhead. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 62–76. Springer, Heidelberg, August 1991.

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

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

BGI16a. Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Heidelberg, August 2016.

BGI16b. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

BGI17. Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193. Springer, Heidelberg, April / May 2017.

BGI+18. Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *ITCS 2018*, volume 94, pages 21:1–21:21. LIPIcs, January 2018.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

BI05. Omer Barkol and Yuval Ishai. Secure computation of constant-depth circuits with applications to database search problems. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 395–411. Springer, Heidelberg, August 2005.

BKS19. Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33. Springer, Heidelberg, May 2019.

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

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

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

CEMT09. James Cook, Omid Etesami, Rachel Miller, and Luca Trevisan. Goldreich's one-way function candidate and myopic backtracking algorithms. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 521–538. Springer, Heidelberg, March 2009.

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

CG97. Benny Chor and Niv Gilboa. Computationally private information retrieval (extended abstract). In *29th ACM STOC*, pages 304–313. ACM Press, May 1997.

CGKS95. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, October 1995.

CM21. Geoffroy Couteau and Pierre Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 842–870. Springer, Heidelberg, October 2021.

COS+22. Ilaria Chillotti, Emmanuela Orsini, Peter Scholl, Nigel Paul Smart, and Barry Van Leeuwen. Scooby: Improved multi-party homomorphic secret sharing based on FHE. *SCN 2022*, 2022. https://eprint.iacr.org/2022/862.

Cou19. Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 473–503. Springer, Heidelberg, May 2019.

DFH12. Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 54–74. Springer, Heidelberg, March 2012.

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

DHRW16. Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Heidelberg, August 2016.

FGJI17. Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith III. Homomorphic secret sharing from paillier encryption. In *Provable Security - 11th International Conference, ProvSec 2017, Xi'an, China, October 23-25, 2017, Proceedings*, volume 10592, pages 381–399, 2017.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

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

GJ11. Anna Gál and Jing-Tang Jang. The size and depth of layered boolean circuits. *Information Processing Letters*, 111(5):213–217, 2011.

GM82. Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM STOC*, pages 365–377. ACM Press, May 1982.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

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

Har77. Lawrence H. Harper. An log lower bound on synchronous combinational complexity. 1977.

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

JLS21. Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

JLS22. Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Heidelberg, May / June 2022.

Kil92. Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

KO97. Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th FOCS*, pages 364–373. IEEE Computer Society Press, October 1997.

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

MST03. Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On e-biased generators in NC0. In *44th FOCS*, pages 136–145. IEEE Computer Society Press, October 2003.

NN01. Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *33rd ACM STOC*, pages 590–599. ACM Press, July 2001.

OSY21. Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Heidelberg, October 2021.

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

Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

RS21. Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 687–717, Virtual Event, August 2021. Springer, Heidelberg.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.