Multi-party Homomorphic Secret Sharing and Sublinear MPC from Sparse LPN

Quang Dao[1(✉)], Yuval Ishai[2], Aayush Jain[1], and Huijia Lin[3]

[1] Carnegie Mellon University, Pittsburgh, USA
{qvd,aayushja}@andrew.cmu.edu
[2] Technion, Haifa, Israel
yuvali@cs.technion.ac.il
[3] University of Washington, Seattle, USA
rachel@cs.washington.edu

Abstract. Over the past few years, homomorphic secret sharing (HSS) emerged as a compelling alternative to fully homomorphic encryption (FHE), due to its feasibility from an array of standard assumptions and its potential efficiency benefits. However, all known HSS schemes, with the exception of schemes built from FHE or indistinguishability obfuscation (iO), can only support two parties.

In this work, we give the first construction of a *multi-party* HSS scheme for a non-trivial function class, from an assumption not known to imply FHE. In particular, we construct an HSS scheme for an *arbitrary* number of parties with an *arbitrary* corruption threshold, supporting evaluations of multivariate polynomials of degree $\log / \log \log$ over arbitrary finite fields. As a consequence, we obtain a secure multiparty computation (MPC) protocol for any number of parties, with (slightly) *sub-linear* per-party communication of roughly $O(S/\log \log S)$ bits when evaluating a layered Boolean circuit of size S.

Our HSS scheme relies on the *sparse* Learning Parity with Noise (LPN) assumption, a standard variant of LPN with a sparse public matrix that has been studied and used in prior works. Thanks to this assumption, our construction enjoys several unique benefits. In particular, it can be built on top of *any* linear secret sharing scheme, producing noisy output shares that can be error-corrected by the decoder. This yields HSS for low-degree polynomials with optimal download rate. Unlike prior works, our scheme also has a low computation overhead in that the per-party computation of a constant degree polynomial takes $O(M)$ work, where M is the number of monomials.

1 Introduction

Homomorphic secret sharing (HSS) [19] is the secret sharing analogue of homomorphic encryption [37,51], which supports local evaluation of functions on shares of secret inputs. A standard N-party t-private secret sharing scheme randomly splits an input x into N shares, $(x_1, \ldots, x_N)$, such that any subset of t shares reveals nothing about the input. An HSS scheme additionally supports computations on shared inputs by means of local computations on their

H. Handschuh and A. Lysyanskaya (Eds.): CRYPTO 2023, LNCS 14082, pp. 315–348, 2023.
https://doi.org/10.1007/978-3-031-38545-2_11

shares. More concretely, there is a local evaluation algorithm Eval and reconstruction algorithm Rec satisfying the following homomorphism requirement. Given a description of a function f, the algorithm $\mathsf{Eval}(f, x_j)$ maps an input share x_j to a corresponding output share y_j such that $\mathsf{Rec}(y_1, \ldots, y_m) = f(x)$. To avoid trivial solutions,[1] the HSS output shares should be *compact* in the sense that their length depends only on the output length of f and the security parameter, and hence the reconstruction time does not grow in the function size. HSS enables private outsourcing of computation to multiple non-colluding servers. It also has applications to secure multiparty computation (MPC) with sublinear communication [19,27,29], multi-server private information retrieval (PIR) and secure keywords search [18,39,53], generating correlated pseudorandomness [15,16], and much more.

The work of Boyle, Gilboa and Ishai [19] gave the first nontrivial example of a 2-party HSS scheme without FHE. Their scheme supports the class of polynomial-size branching programs (which contains NC^1) and is based on the Decisional Diffie-Hellman (DDH) assumption. A series of followup works have extended their result, improving efficiency [17,20,22], and diversifying the underlying assumptions to Decision Composite Residuosity (DCR) [33,50,52] or assumptions based on class groups of imaginary quadratic fields [1]. For more limited function classes, which include constant-degree polynomials, 2-party HSS can be based on different flavors of the Learning Parity with Noise (LPN) assumption [16,29]. However, when it comes to the general setting of HSS with $N \geq 3$ parties, constructions have been lacking, with the only known solutions relying on either FHE [11,12,21,26,32,48] or Indistinguishability Obfuscation (iO) [18].[2]

The same "multi-party barrier" exists when it comes to the construction of sublinear-communication MPC protocols, the goal is to achieve (per-party) communication cost that is sublinear in the size of the circuit being computed. Until the DDH-based construction of HSS [19], this could only be achieved using FHE. It is easy to see that an N-party $(N-1)$-private HSS for a function class $\mathcal{F}$ directly implies an MPC protocol for functions in $\mathcal{F}$ with communication depending only on the input and output lengths. Thus, all 2-party HSS schemes in previous works immediately yield 2-party low-communication MPC for low-depth computations (log- or log log-depth). Furthermore, these protocols can be extended to handle general layered circuits with a communication cost sublinear in the circuit size (by a log or log log factor). Unfortunately, when it comes to general multiparty settings, with up to $N-1$ corruption, the only known solutions again rely on FHE or $i\mathcal{O}$.

Motivated by the state-of-the-art, we ask:

[1] A trivial solution is letting the output shares be (f, x_j) and Rec reconstruct x from the shares and then compute f. However, this solution is uninteresting since it is not useful.

[2] The work of [18] builds 2-party HSS for general polynomial-sized computation from subexponentially secure iO and one-way functions. Their construction can be extended to the multiparty setting.

Can we have general N-party t-private HSS for useful classes of functions, and sublinear communication MPC for general number of parties, without FHE or iO?

1.1 Our Results

In this work, based on the sparse LPN assumption (described shortly), we construct general N-party t-private HSS for log log-depth arithmetic circuits, and more generally, for the class of multivariate polynomials with log / log log degree and a polynomial number of monomials. Our HSS natively supports arithmetic computation over arbitrary field $\mathbb{F}_q$ (assuming sparse LPN over $\mathbb{F}_q$). It also enjoys concrete efficiency. In particular, the server computation overhead can be made constant (independent of the security parameter) when evaluating constant degree polynomials, and shares of multiple outputs can be packed together to achieve optimal download rate [35]. As an application of our HSS, we obtain the first sublinear-communication MPC for general layered circuits and arbitrary number of parties without relying on FHE or $i\mathcal{O}$. We now describe our results in more detail.

Sparse LPN. Given two sparsity parameters $k = \omega(1) \in \mathbb{N}$ and $\delta \in (0, 1)$, the (k, δ)-sparse LPN assumption over a finite field $\mathbb{F}_q$ states that the following distributions are computationally indistinguishable:

$$(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T \mod q) \approx_c (\mathbf{A}, \mathbf{r}) \ , \text{ where } \boldsymbol{A} \in \mathbb{F}_q^{n\times m},\ \mathbf{s} \leftarrow \mathbb{F}_q^n,\ \boldsymbol{e} \in \mathbb{F}_q^m, \mathbf{r} \leftarrow \mathbb{F}_q^m \ .$$

The public matrix $\boldsymbol{A} \in \mathbb{F}_q^{n\times m}$ is k-sparse, meaning that each column is sampled randomly subject to having Hamming weight exactly k, while the error vector $\mathbf{e}$ is $n^{-\delta}$-sparse in the sense that each coordinate e_i is random non-zero with probability $1/n^\delta$ and 0 otherwise. This work relies on the *sparse LPN assumption* that the above indistinguishability holds for every super-constant k, every constant $\delta \in (0, 1)$, every prime modulus q (potentially exponentially large in λ), and any polynomial number m of samples. See Assumption 4.1 for the precise formulation. In fact, in this work it suffices to require the above indistinguishability to hold for *some* $\delta > 0$ (though we believe that the assumption should hold for any constant $\delta \in (0, 1)$).

Variants of this assumption in the binary field $\mathbb{F}_2$ have been proposed and studied for at least a couple of decades in average-case complexity (see works such as [2,3,10,30,34,40,45,47]). The work of [6] generalized the assumption to large fields $\mathbb{F}_q$. Both of these variants have been used in a number of works (see for example [2,4,7,42]). The related assumption of local PRGs [40] has also been used in a number of works including the recent construction of program obfuscation scheme [44]. Comparing with previous variants, our assumption is relatively conservative in two aspects. First, we consider public matrices that are $(k = \omega(1))$-sparse, instead of constant sparse $k = O(1)$. In fact, for our constructions of HSS and sublinear communication MPC, it suffices to set $k = \mathsf{poly}(\log \lambda)$. Second, the error-rate $1/n^\delta$ can be an arbitrary inverse polynomial,

whereas for some application such as PKE [4] we require δ to be greater than some fixed constant.

The work of [4] showed how to construct PKE from sparse LPN over $\mathbb{F}_2$ with constant sparsity $k = 3$, sample complexity $n^{1.4}$ and error probability $o(n^{-0.2})$. Their scheme could be naturally extended to work with the variant of the assumption for a fairly general choice of parameters. In particular, they could work with any choice of constant $k \geq 3$, assuming a sample complexity of $m = n^{1+(k/2-1)(1-\delta)}$ for $\delta > 0$, where the noise probability should be $o(n^{-\delta})$. In our case, k is set to be $\omega(1)$ (so n^k is super-polynomial), and our sample complexity is only polynomial in n. For these parameters the noise probability implying PKE through [4] is smaller than any inverse polynomial, while for us, the noise probability could be $n^{-\delta}$ for any $\delta > 0$. Therefore, to the best of our knowledge, our parameters are not known to imply PKE. We survey cryptanalysis of the sparse LPN problem, and give more details on the PKE scheme, in the full version.

General N-party t-private HSS Scheme. Assuming sparse LPN, we present a construction of HSS schemes for general number of parties N and privacy threshold t. Our schemes support computing functions represented by multivariate polynomials with degree $O(\log \lambda / \log\log \lambda)$ and polynomial number of monomials; in particular, this class of functions contains $O(\log\log)$-depth arithmetic circuits. However, similar to the DDH-based HSS construction of [19], our schemes have a noticeable correctness error, which can be made as small as any inverse polynomial, at the cost of worse efficiency.

Theorem 1.1 (Multi-party HSS, informal). *Assume sparse LPN. For any number of parties $N \geq 2$, privacy threshold $t < N$, modulus q, error probability $\epsilon = 1/\mathsf{poly}(\lambda)$, there is a N-party, t-private HSS with correctness error ϵ for the following class of functions:*

- Function Class $\mathcal{P}(\mathbb{F}_q, D, M)$*: multivariate polynomials over the finite field $\mathbb{F}_q$ with degree $D = O(\log \lambda / \log\log \lambda)$ and number of monomials $M = \mathsf{poly}(\lambda)$.*

The reconstruction of the above HSS scheme is linear. Furthermore, the scheme can be modified to have compact (but non-linear) reconstruction and negligible error rate.

Previously, sparse LPN with specific parameters was used to build public-key encryption (PKE) through the classic work of [4]. However, as remarked above, our parameters implying HSS are not known to imply PKE. Therefore, we obtain the first multi-party HSS scheme for useful classes of functions from a plausibly mini-crypt assumption. In contrast, previous (2-party) HSS schemes were either based on LWE, on various number theoretic assumptions (DDH/DCR/QR), or on standard LPN (with dense public matrix) that required the error rate to be below $n^{-0.5}$; all of these assumptions are known to imply PKE. See Fig. 1 for details.

Besides accommodating general N and t, our construction enjoys several other desirable features. First, thanks to the fact that the sparse LPN assumption

Assumptions	(N, t)	Function Class	Error
DDH [17,19,20], DCR [33]	$(2, 1)$	Branching programs (NC^1)	$1/\operatorname{poly}$
LWE [22]	$(2, 1)$	Branching programs (NC^1)	negl
DCR [50,52]	$(2, 1)$	Branching programs (NC^1)	negl
Class Groups [1]	$(2, 1)$	Branching programs (NC^1)	negl
LPN [16]	$(2, 1)$	Constant-degree polynomials	none
Quasi-poly LPN [29]	$(2, 1)$	Loglog-depth circuits	none
Degree-k Homomorphic Encryption [43,46]	$\left(\lfloor \frac{dt}{k+2} \rfloor, t\right)$	Degree-d polynomials	none [a]
Unconditional (Shamir-based) [35]	$(dt + 1, t)$	Degree-d polynomials	none
iO and OWF [18]	$(\star, \star)$	Circuits ($\mathsf{P}/\operatorname{poly}$)	none
FHE [21,32] [b]	$(\star, \star)$	Circuits ($\mathsf{P}/\operatorname{poly}$)	negl
Sparse LPN (Ours)	$(\star, \star)$	Loglog-depth circuits	$1/\operatorname{poly}$

a reconstruction is non-linear

b relies on multi-key FHE schemes that can be based on "circular-secure" LWE

Fig. 1. Comparison between existing N-party, t-private HSS schemes and ours. The reconstruction process is linear unless stated otherwise.

"arithmetize" to arbitrary field $\mathbb{F}_q$, our HSS schemes natively support evaluating these polynomials (and arithmetic circuits) over arbitrary field $\mathbb{F}_q$. Second, our construction can also accommodate general reconstruction threshold $t < t' \leq N$, namely, how many output shares are needed in order to reconstruct the output. Having a smaller reconstruction threshold are useful in certain applications, for instance, it implies fault tolerance to server failures in the scenario of outsourcing computation to multiple servers via HSS. Furthermore, our schemes have constant server-computation overhead when computing low degree polynomials and optimal download rate, which we expand in detail later.

Sublinear Communication MPC for Any Number of Parties. Using our HSS construction, we circumvent the "circuit-size barrier" for general MPC, for the first time, without restricting the number of parties N or the function classes, nor using FHE or $i\mathcal{O}$. We construct such protocols where the communication cost of each party is sublinear in the size S of the Boolean layered circuit being computed, roughly by a factor of $\log \log S$ (plus other lower order terms).

Theorem 1.2 (Sublinear MPC, informal). *Assume sparse LPN and the existence of an oblivious transfer protocol. Then, for any $\kappa(\lambda) \in \omega(1)$ and any number of parties N, there exist N-party MPC protocols tolerating up to $(N-1)$ semi-honest corruptions that can evaluate Boolean layered circuits of size S, depth D, and width W, with per-party communication*

$$O(\kappa \cdot S/\log \log S) + D \cdot S^{o(1)} \cdot \mathsf{poly}(\lambda, N) + W \cdot \mathsf{poly}(\log N, \log \lambda)/N.$$

Besides sparse LPN, our sublinear-communication MPC also (inevitably) needs to rely on an Oblivious Transfer (OT) protocol. The latter can be based on standard LPN with noise rate below $n^{-0.5}$ [2,31] or a specific sparse LPN-type assumption [4].[3] In summary, sublinear-communication MPC can be obtained using only assumptions in the LPN family.

Finally, we note that by an existing compiler due to Naor and Nissim [49], we can upgrade our MPC protocols to be maliciously secure while preserving per-party sublinear communication cost, assuming the existence of Collision-Resistant Hash (CRH) functions. Again, CRH can be constructed from standard LPN with low-noise rate $\log^2(n)/n$ [24].

Low Server Computation Overhead. If assuming stronger variants of sparse LPN assumption where the public matrix is constant-sparse,[4] i.e., $k = O(1)$, we can slightly adapt the evaluation procedure of our HSS construction, so that, the computation overhead of each party/server for computing *constant-degree* polynomials represented as a sum of monomials is only a constant. More precisely, to compute a single degree d monomial over $\mathbb{F}_q$, the local homomorphic evaluation procedure can be represented by a degree d arithmetic circuit over $\mathbb{F}_q$ of size $O((k+1)^d)$. Next, homomorphic addition of the outputs of t monomials involves only t addition over $\mathbb{F}_q$. Therefore, when both k and d are constants, the overhead is at most $O((k+1)^d/d)$ (i.e., the ratio between the server cost and the cost of computing a single monomial) a constant. In comparison, almost all previous HSS schemes (tolerating $N-1$ corruption) have a server computation overhead proportional to the security parameter $\mathsf{poly}(\lambda)$ [1,22,29,33,50,52]; the only exception is using FHE [38] with polylogarithmic overhead, which implies HSS with $\mathsf{poly}(\log \lambda)$ overhead.

We remark that HSS for low-degree polynomials is well-motivated by a variety of applications, for instance, multi-server private information retrieval, for computing inner product between two integer-valued vectors (a degree-2 function) which is a measure of correlation, and for computing intersection of d sets where each set is represented by a characteristic vector in $\mathbb{F}_2^\ell$, and intersection can be computed by ℓ instances of a degree-d monomial over $\mathbb{F}_2$. See [35,43,46] for more examples.

Simple Reconstruction and Optimal Download Rate. In fact, our HSS is also "compatible" with an arbitrary *multi-secret sharing scheme* LMSS. This allows us to achieve much better download rate[5] by packing many function evaluations into a single set of output shares. In particular, by plugging in the multi-secret Shamir sharing [36], we achieve a rate of $1-t/N$, which matches the best possible rate for information-theoretic HSS. In fact, this also applies to computational

[3] Namely, the PKE constructed in [4] can be directly transformed into a semi-honest OT.

[4] In such a setting, we shall use public matrices from specific distributions instead of being uniform. See Remark 4.2 for a discussion.

[5] The download rate is the ratio of the output size over the sum of all output share sizes (for details, see [35]).

HSS with linear reconstruction, or where the output share size is independent of the computational security parameter.[6]

Theorem 1.3 (General Linear Output Shares, informal). *For any field $\mathbb{F}_q$, assume sparse LPN over $\mathbb{F}_q$. For any $N \geq 2$, $t < N$, $\epsilon = 1/\mathsf{poly}(\lambda)$, and any N-party, t-private linear secret sharing scheme LSS, there is an N-party, t-private HSS with correctness error ϵ for the same function class as in Theorem 1.1 satisfying the following properties:*

- *the output shares are LSS secret shares of the output y with probability $1 - \epsilon$ (and LSS secret shares of some wrong value with probability ϵ).*
- *using an appropriate LMSS, the output shares can be packed together to achieve download rate $1 - t/N$.*

The above should be compared with the $1 - Dt/N$ rate of the (perfectly correct) *information-theoretic* construction from [35], which is in fact optimal for HSS in which *both* the sharing and the reconstruction are linear. To the best of our knowledge, the only other computationally secure HSS scheme with $(1-t/N)$ download rate uses FHE with certain properties. This scheme is sketched in the full version.

1.2 Related Work

2-party sublinear MPC. The work of Boyle, Gilboa, and Ishai [19] showed how to build sublinear 2PC for layered circuits of size S with communication complexity roughly $O(S/\log S)$, under the DDH assumption. Following this template, later works showed that we can replace DDH with various other assumptions such as DCR [33,50,52], poly-modulus LWE [22], or class group assumptions [1]. More recently, Couteau and Meyer [29] showed that assuming the quasi-polynomial hardness of (dense) LPN, we can have 2PC with sublinear communication complexity roughly $O(S/\log\log S)$. Finally, in the correlated randomness model with polynomial storage, Couteau constructed information theoretically secure MPC protocols with communication complexity O(S/log log S) [27].

Beyond 2 parties. In a very recent and independent work, Boyle, Couteau and Meyer [14] constructed the first sublinear MPC protocols for $N \geq 3$ parties from assumptions that are not known to imply FHE. This includes a 3-party protocol from a combination of a variant of the (dense) LPN assumption and *either* DDH or QRA, as well as a 5-party protocol additionally assuming DCR and a local PRG. In contrast, we obtain a sublinear MPC protocol for *any* number of parties N, based entirely on variants of the LPN assumption (sparse LPN and OT, which is implied from low-noise dense LPN).

Our technical approach is very different from that of [14]. The results of [14] are based on a novel compiler that obtains a sublinear N-party MPC protocol

[6] The latter conditions rule out HSS schemes in which the output shares contain a homomorphic encryption of the output. Such schemes can only achieve good rate when the output size is much bigger than the (computational) security parameter.

from an $(N-1)$-party HSS scheme satisfying an extra "Las-Vegas"[7] correctness property, along with a PIR scheme with special properties. (See [14] for details, and Proposition 1 in [14] for a more general framework.) We cannot use the compiler from [14] to obtain our MPC result (Theorem 1.2), for two reasons: our HSS scheme does not satisfy the extra Las-Vegas property, and (even standard) PIR is not known to follow from any variant of LPN.

Instead, our sublinear MPC protocol follows the blueprint of a similar (2-party) HSS-based construction from [19], adapting it to the lower complexity class supported by our HSS scheme and extending it to cope with a big number of parties. This approach is more direct and simpler than the compiler from [14], thanks to the fact that we can use an N-party (rather than an $(N-1)$-party) HSS scheme to construct N-party sublinear MPC protocols.

Finally, we note that while our MPC protocol inherently has a negligible correctness error, the construction in [14] can leverage HSS schemes with Las-Vegas correctness to yield perfectly correct (3-party or 5-party) MPC protocols [28]. We leave open the possibility of obtaining a Las-Vegas variant of our HSS scheme or a perfectly correct sublinear MPC from sparse LPN and OT.

2 Technical Overview

Our results are facilitated mainly due to structural properties underlying our assumption of *sparse LPN*.

Sparse LPN. We start by recalling the *sparse LPN* assumption. Our assumption states that the following two distributions are computationally indistinguishable:

$$\{\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e_i\}_{i \in [m]} \approx_c \{\boldsymbol{a}_i, u_i\}_{i \in [m]},$$

where $\boldsymbol{a}_i$ are randomly chosen k-sparse vectors over $\mathbb{F}_q^n$ for a prime power q, and m is an arbitrarily chosen polynomial in n. The error e_i is chosen sparsely from a Bernoulli random variable over $\mathbb{F}_q$ with probability of error being $n^{-\delta}$ for a constant $\delta > 0$. On the other hand, $\{u_i\}$ are chosen at random from $\mathbb{F}_q$. In this work, we can work with any $\delta > 0$ and typically consider $k = \omega(1)$ as an appropriately chosen super constant, however the assumption is plausible even when k is chosen to be a constant integer greater than equal to 3 as long as $m = o(n^{k/2})$. Such an assumption will allow our homomorphic secret sharing scheme to support a slightly bigger function class. We discuss the history and cryptanalysis of this assumption in the full version. Our function class consists of multivariate polynomials over $\mathbb{F}_q$, and the sparse LPN assumption we will use to build such an HSS will also be over the same $\mathbb{F}_q$. We now illustrate how this assumption gives rise to a conceptually clean construction of a homomorphic secret sharing scheme.

[7] An HSS scheme has Las-Vegas correctness with error ϵ if each output share can be set to $\perp$ with at most ϵ probability, and if no output share is set to $\perp$ then the shares must always add up to the correct output.

2.1 HSS Construction

We now describe the ideas behind our HSS construction. In this work, we consider the function class $\mathcal{P}(\mathbb{F}_q, D, M)$ which consists of polynomials evaluated on inputs that are vectors of arbitrary polynomial length over $\mathbb{F}_q$. These polynomials are of degree D, and are subject to an upper bound of M on the number of monomials. Looking ahead, we will handle $D = \frac{\log \lambda}{\log \log \lambda}$ and $M = \mathsf{poly}(\lambda)$ where λ is the security parameter. This already lets us evaluate Boolean circuits that are local where the locality[8] is bounded by $D = \frac{\log \lambda}{\log \log \lambda}$ - any circuit that has a locality bounded by D, can be represented by such a polynomial. We refer to the definitions for a homomorphic-secret sharing scheme $\mathsf{HSS} = (\mathsf{Share}, \mathsf{Eval}, \mathsf{Rec})$ in Sect. 3.2.

Template from Boyle et al. Our scheme follows the same high-level template that was first suggested by [19] and has been later adopted in a number of follow-ups such as [1,22,50,52], but introduces a number of important twists. Suppose we want to secret share a vector $\boldsymbol{x} \in \mathbb{F}_q^m = (x_1, \ldots, x_m)$ amongst N parties. We work with a suitable linear secret sharing scheme over $\mathbb{F}_q$. In this overview, we will work with the additive secret sharing for N parties, but it could be any linear secret sharing scheme over $\mathbb{F}_q$ (or its field extensions).

Each party can be handed over the shares of $\boldsymbol{x}$: a party P_ℓ for $\ell \in [N]$ is given shares that are denoted by $[\![x_i]\!]_\ell$ for $i \in [m]$, respectively. This is already enough to build a homomorphic secret sharing scheme supporting linear functions. Namely, parties can locally compute shares of linear functions of $\boldsymbol{x}$ by applying appropriate linear functions over their shares $[\![x_i]\!]_\ell$.

The main ingredient in prior HSS schemes is a method that lets one non-interactively compute share of multiplication of an intermediate computation y with an input symbol x_i. The idea is that one publishes an encryption of the input $\{\mathsf{ct}_{\boldsymbol{s}}(x_i)\}$ and encryptions of the products $\{\mathsf{ct}_{\boldsymbol{s}}(x_i \cdot s_j)\}$, where $\boldsymbol{s} = (s_1, \ldots, s_n)$ is the secret key, using a suitably chosen linearly homomorphic encryption scheme (such schemes can typically be instantiated from any of the LWE/DDH/DCR/QR assumptions). Since we are encrypting functions of the secret key inside the ciphertext, the encryption scheme must also be KDM secure (or we must assume it is KDM secure).

These encryptions are given to all parties. Along with these encryptions and the shares of the input $[\![x_i]\!]_\ell$, each party P_ℓ receives a share $[\![x_i \cdot s_j]\!]_\ell$ of the product $x_i s_j$. The key step is a procedure that allows one to start with a share $[\![y]\!]_\ell$ for an intermediate computation y and shares $[\![y \cdot s_j]\!]_\ell$ of products $y \cdot s_j$ and compute not only a share of $[\![y \cdot x_i]\!]_\ell$ for any input x_i but also shares of the form $[\![y \cdot x_i \cdot s_j]\!]_\ell$. This step leverages structural properties of the linearly homomorphic encryption scheme. Typically in such settings, one homomorphically computes on the ciphertext by "multiplying" $\mathsf{ct}_{\boldsymbol{s}}(x_i)$ with $[\![y]\!]_\ell$. The resulting ciphertext is then "decrypted" in a distributed fashion using the secret-shares of the form $[\![y \cdot s_j]\!]_\ell$, assuming that the decryption is almost linear. This produces

[8] The locality of any Boolean circuit is the number of input bits it depends on.

shares of $[\![y \cdot x_i]\!]_\ell$. The shares of $[\![y \cdot x_i \cdot s_j]\!]_\ell$ can be computed by starting with $\mathsf{ct}_{\boldsymbol{s}}(x_i s_j)$ instead.

Which linearly homomorphic encryption one chooses can present different sets of challenges for realizing the above step. [19,50] relied on DDH/Pallier based encryption. Since the ambient space of shares is over some field, whereas the encryption consists of group elements, this step include some operations done over the groups followed by a "distributed discrete-log" step that works specifically for two parties. In LWE based schemes such as [22], the ciphertexts live in the same space as that of the shares. The issue is that while the ciphertexts are almost linear in the secret, they have a low-norm error. The authors suggest a rounding based idea that was inspired by earlier works on homomorphic encryption [23,25] that for some (not so) coincidental reason works specifically for two parties. Our main approach consists of devising a suitable linearly homomorphic encryption that sits naturally over the field $\mathbb{F}_q$, and does not suffer from the issues that prevented scaling of previous ideas beyond two parties.

Suitable Linearly Homomorphic Encryption. The main issue with prior linear homomorphic encryption schemes that restricts constructions to two parties is that they don't work naturally with the linear secret sharing scheme. As a result, special share conversion methods have to be devised (which seem to be stuck at two parties). It is instructive to ask what properties a linear homomorphic encryption could satisfy so that it works more naturally with the linear secret sharing scheme.

To this end, consider the following (broken) encryption scheme that encrypts input x_i as $\mathsf{ct}_{\boldsymbol{s}}(x_i) = (\boldsymbol{a}_i, b_i = \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + x_i)$ where $\boldsymbol{s}$ is a vector of $\mathbb{F}_q^n$ and $\boldsymbol{a}_i \leftarrow \mathbb{F}_q^n$ is randomly chosen. Similarly, we have $\mathsf{ct}_{\boldsymbol{s}}(x_i s_j) = (\boldsymbol{a}_{i,j}, b_{i,j} = \langle \boldsymbol{a}_{i,j}, \boldsymbol{s} \rangle + x_i s_j)$. Such an encryption scheme is both linearly homomorphic and has a linear decryption function over $\mathbb{F}_q$. On the other hand, it is obviously not secure: one could find the secret $\boldsymbol{s}$ by solving a properly constructed linear equation system.

But, for the time being assume that the scheme was secure. If this were true, then this will give rise to a homomorphic secret sharing scheme supporting corruption patterns governed by any linear secret sharing scheme over $\mathbb{F}_q$, thanks to it being linearly homomorphic over $\mathbb{F}_q$ and having linear decryption over $\mathbb{F}_q$. Indeed, observe that

$$b_i \, [\![y]\!]_\ell - \langle \boldsymbol{a}_i, ([\![y s_1]\!]_\ell, \ldots, [\![y s_n]\!]_\ell) \rangle = [\![x_i \cdot y]\!]_\ell \tag{1}$$

$$b_{i,j} \, [\![y]\!]_\ell - \langle \boldsymbol{a}_{i,j}, ([\![y s_1]\!]_\ell, \ldots, [\![y s_n]\!]_\ell) \rangle = [\![x_i \cdot y \cdot s_j]\!]_\ell \tag{2}$$

LPN-Based Linearly Homomorphic Encryption. While the above proposal would work, as described before, it is obviously not secure. To fix the security issue, one could leverage an encryption scheme based on the *standard LPN assumption.* We could instead have $\mathsf{ct}_{\boldsymbol{s}}(x_i) = (\boldsymbol{a}_i, b_i = \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + x_i + e_i)$ and $\mathsf{ct}_{\boldsymbol{s}}(x_i s_j) = (\boldsymbol{a}_{i,j}, b_{i,j} = \langle \boldsymbol{a}_{i,j}, \boldsymbol{s} \rangle + x_i s_j + e_{i,j})$ where e_i and $e_{i,j}$ are chosen from the generalized Bernoulli random variables $\mathsf{Ber}(\mathbb{F}_q, \eta)$ where $\eta(n)$ is chosen to be a small inverse polynomial $n^{-\delta}$. The resulting scheme is now secure by the LPN assumption, it is also linearly homomorphic and has a linear decryption over $\mathbb{F}_q$. Although, the

decryption has a small probability correctness error due to noise. The problem we now face is correctness of the output.

We can observe that if one is initially given $[\![x_i]\!]_\ell$ and $[\![x_i \cdot s_j]\!]_\ell$, as one computes shares for degree two computations $x_{i_1} \cdot x_{i_2}$, Eq. 1 instead yields noisy shares $\langle\!\langle x_{i_1} \cdot x_{i_2} \rangle\!\rangle_\ell$ and $\langle\!\langle x_{i_1} \cdot x_{i_2} \cdot s_j \rangle\!\rangle_\ell$. Here by "noisy" we don't mean that the shares of individual parties are corrupted, but rather that, with some small probability the shares reconstruct to something else other than the desired computation (but they are still consistent secret sharing of some "noisy" output). Each computed share can be corrupted with probability η due to the LPN noise. Moreover, as one evaluates further to compute degree three terms, the noise increases further. To compute degree three shares of the form $\langle\!\langle x_{i_1} \cdot x_{i_2} \cdot x_{i_3} \rangle\!\rangle_\ell$, the noise probability could already be overwhelming. This is because due to Eq. 1,

$$b_{i_3} \langle\!\langle x_{i_1} x_{i_2} \rangle\!\rangle_\ell - \langle \boldsymbol{a}_i, (\langle\!\langle x_{i_1} x_{i_2} s_1 \rangle\!\rangle_\ell, \ldots, \langle\!\langle x_{i_1} x_{i_2} s_n \rangle\!\rangle_\ell) \rangle = \langle\!\langle x_{i_1} x_{i_2} x_{i_3} \rangle\!\rangle_\ell .$$

Thus, each conversion is a function of one LPN sample and n shares derived in the previous layer. The probability of having no noise in the reconstructed output is roughly the probability that all the shares derived in the previous layer are non-noisy and the LPN sample used in that layer has no noise. This probability is roughly $(1-\eta)^{O(n)}$ assuming that the errors are independent. As $\eta \gg \frac{1}{n}$, this probability is already negligible.

Sparse LPN for Error Control. We can observe that in Eq. 1 above (now with noisy shares),

$$b_{i_3} \langle\!\langle x_{i_1} x_{i_2} \rangle\!\rangle_\ell - \langle \boldsymbol{a}_i, (\langle\!\langle x_{i_1} x_{i_2} s_1 \rangle\!\rangle_\ell, \ldots, \langle\!\langle x_{i_1} x_{i_2} s_n \rangle\!\rangle_\ell) \rangle = \langle\!\langle x_{i_1} x_{i_2} x_{i_3} \rangle\!\rangle_\ell ,$$

if $\boldsymbol{a}_i$ was only k-sparse, where k is a parameter that could be a constant or slightly super-constant, the error build up will be manageable. The probability that the share is non-noisy is can now be lower-bounded by $1-(k+1)\eta$. This is because this equation now depends only on $k+1$ noisy shares derived in the previous layer and one LPN sample, both with noise rate η.

Going inductively, the shares at level D for computing a degree D monomial are non-noisy with probability at least $1 - O((k+1)^D \eta)$. If one further adds M such degree D monomials to compute the polynomial of desired form the resulting shares are non-noisy with probability is at least $1 - O(M(k+1)^D \eta)$. We can make sure that this probability is $1 - O(\frac{1}{\lambda})$ if $M(k+1)^D \eta$ is kept smaller than $\frac{1}{\lambda}$. If M is some polynomial in λ, $D = \frac{\log \lambda}{\log \log \lambda}$, and $\eta = n^{-\delta}$ for some constant $\delta > 0$, we can set $k = \log^{O(1)} \lambda$ and n as some other polynomial in λ. More details of our HSS scheme can be found in Sect. 5.1.

Summing up. To sum up, one would compute

$$\mathsf{ct}_{\boldsymbol{s}}(x_i) = (\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e_i + x_i), \tag{3}$$

where $\boldsymbol{a}_i$ is chosen to be a random sparse vector as in the distribution specified by the sparse LPN assumption, and e_i is generated as a sparse noise. $\{\mathsf{ct}_{\boldsymbol{s}}(x_i s_j)\}_{i,j}$

are generated analogously. Since our assumption works naturally over the field $\mathbb{F}_q$ one could use any linear secret sharing scheme over $\mathbb{F}_q$. One can then evaluate any function in $\mathcal{P}(\mathbb{F}_q, D, M)$. For any function f in the function class, at the end of the evaluation each party gets a noisy share $\langle\langle f(x_1, \ldots, x_m)\rangle\rangle_\ell$. With all but a small inverse polynomial probability, these shares reconstruct to $f(\boldsymbol{x})$ using the same linear reconstruction that is used for the base secret sharing scheme.

2.2 Arguing KDM Security

One issue that we have not discussed thus far is that in our HSS scheme, one gives out encryptions that are dependent on the key $\boldsymbol{s}$. Namely, all parties not only get encryptions of the input $\mathsf{ct}_{\boldsymbol{s}}(x_i)$, but also encryptions $\mathsf{ct}_{\boldsymbol{s}}(x_i s_j)$ of the products $x_i s_j$. Therefore, we need to argue that KDM security follows from sparse LPN. Note that indeed if $\mathsf{ct}_{\boldsymbol{s}}(x_i s_j)$ were encrypted using the standard LPN assumption, namely by setting $\mathsf{ct}_{\boldsymbol{s}}(x_i s_j) = (\boldsymbol{a}_{i,j}, \langle \boldsymbol{a}_{i,j}, \boldsymbol{s}\rangle + e_{i,j} + x_i s_j)$, where $\boldsymbol{a}_{i,j}$ is chosen randomly over $\mathbb{F}_q^n$, then such KDM security holds directly from LPN. The idea is that one can "simulate" such an encryption from an LPN sample $(\boldsymbol{a}', b' = \langle \boldsymbol{a}', \boldsymbol{s}\rangle + e)$ as follows. We can simply set $\boldsymbol{a}_{i,j} = \boldsymbol{a}' - \underbrace{(0, \ldots, 0, x_i, 0, \ldots 0)}_{x_i \text{ at } j^{th} \text{ coordinate}} = \boldsymbol{a}' - x_i \cdot \boldsymbol{v}_j$ for the j^{th} unit vector $\boldsymbol{v}_j$, and $b_{i,j} = b'$. Observe that $b' = \langle \boldsymbol{a}', \boldsymbol{s}\rangle + e = \langle \boldsymbol{a}_{i,j}, \boldsymbol{s}\rangle + x_i s_j + e$. Since $\boldsymbol{a}'$ is chosen at random, the distribution of $\boldsymbol{a}_{i,j}$ is also identically random even given x_i.

The above simulation strategy fails to work when $\boldsymbol{a}_{i,j}$ are exactly k-sparse for some k. This is because the vector $\boldsymbol{a}_{i,j}$ that is used to construct $\mathsf{ct}_{\boldsymbol{s}}(x_i \cdot s_j)$ might actually be distinguishable from the distribution of $\boldsymbol{a}' - x_j \boldsymbol{v}_j$. Not only there could be a difference in the number of non-zero coordinates, this could also leak out x_i (by observing the value at of $\boldsymbol{a}_{i,j}$ formed this way at the j^{th} coordinate).

We modify slightly the distribution of the coefficient vectors $\boldsymbol{a}_{i,j}$ used to generate $\mathsf{ct}_{\boldsymbol{s}}(x_i \cdot s_j)$ so that one could prove KDM security under sparse LPN assumption. Below we sketch the main ideas assuming q is a prime power greater than 2.

Modified Distribution. In the actual scheme in Sect. 5, we encrypt the vector $\boldsymbol{x}$ as $\mathsf{ct}_{\boldsymbol{s}}(x_i) = (\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s}\rangle + e_i + x_i)$ where $\boldsymbol{a}_i$ are exactly k-sparse. However, to encrypt the products $x_i s_j$, we compute $\mathsf{ct}_{\boldsymbol{s}}(x_i s_j) = (\boldsymbol{a}_{i,j}, \langle \boldsymbol{a}_{i,j}, \boldsymbol{s}\rangle + e_{i,j} + x_{i,j})$ where $\boldsymbol{a}_{i,j}$ are chosen differently. They are chosen to be $2k-1$ sparse with the constraint that the j^{th} coordinate of $\boldsymbol{a}_{i,j}$ is non-zero. This constraint enables us to prove security from sparse LPN as long as $q > 2$.

Our main idea is that such a sample $\boldsymbol{a}_{i,j}, b_{i,j}$ can be simulated from sufficiently (polynomially) many samples of sparse LPN with sparsity k. Say we have two samples of the form $\boldsymbol{c}_1, d_1$ and $\boldsymbol{c}_2, d_2$ such that $d_i = \langle \boldsymbol{c}_i, \boldsymbol{s}\rangle + e_i$ for $i \in \{1, 2\}$. Additionally, $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are non-zero at the j^{th} coordinate and that is the only coordinate at which both $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are non-zero. Any pair of samples will satisfy this property with an inverse polynomial probability provided

k is reasonably small. We sample a random non-zero field element r, and two non-zero elements $\mu_1, \mu_2 \in \mathbb{F}_q$ so that $\mu_1 c_{1,j} + \mu_2 c_{2,j} = r + x_i$. Computing such non-zero μ_1 and μ_2 requires that $q > 2$. Indeed if $q = 2$, there is only one choice for μ_1 and μ_2 and then our condition $\mu_1 c_{1,j} + \mu_2 c_{2,j} = r + x_i$ may not hold. Now let $\boldsymbol{\alpha} = \mu_1 \boldsymbol{c}_1 + \mu_2 \boldsymbol{c}_2$, and set $\boldsymbol{a}_{i,j} = \boldsymbol{\alpha} - x_i \boldsymbol{v}_j$. Our desired sample then becomes $\mathsf{ct}_{\boldsymbol{s}}(x_i s_j) = (\boldsymbol{a}_{i,j}, b_{i,j} = \mu_1 d_1 + \mu_2 d_2)$. Observe that $b_{i,j} = \mu_1 \langle \boldsymbol{c}_1, \boldsymbol{s} \rangle + \mu_2 \langle \boldsymbol{c}_2, \boldsymbol{s} \rangle + \mu_1 e_1 + \mu_2 e_2$. As $\boldsymbol{a}_{i,j} = \mu_1 \boldsymbol{c}_1 + \mu_2 \boldsymbol{c}_2 - x_i \boldsymbol{v}_j$, we have that $b_{i,j} = \langle \boldsymbol{a}_{i,j}, \boldsymbol{s} \rangle + (\mu_1 e_1 + \mu_2 e_2) + x_i s_j$.

Note that the error $\mu_1 e_1 + \mu_2 e_2$ is still sparse (with noise rate close to 2η); our remaining task is to show that $\boldsymbol{a}_{i,j}$ has the right distribution. This follows from the following argument. Since $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ have disjoint support aside from the j^{th} coordinate, the distribution of $\boldsymbol{a}_{i,j}$ on coordinates not equal to j is identical to a random $(2k-2)$-sparse vector. On the other hand, at the j^{th} coordinate $\boldsymbol{a}_{i,j}$ is set to be equal to r, which is random non-zero.

When $q = 2$, we are not able to prove KDM security of our distribution under (exactly) k-sparse LPN. On the other hand, relying on a related assumption we can indeed show KDM security. In this assumption, the samples will consist of two kinds of coefficient vectors $\boldsymbol{a}_i$: with half probability, $\boldsymbol{a}_i$ will be k-sparse, otherwise it will be $(k-1)$-sparse. We refer to Sect. 4.1 for more details.

2.3 Sublinear MPC Construction

We can leverage our homomorphic secret sharing scheme to build a sublinear MPC protocol for (Boolean) layered circuits. Here too, our result follows the main conceptual outline suggested by [19], with a number of low-level, yet important, differences in the implementation. The differences in implementation come from three sources: handling arbitrary number of parties, dealing with restricted function classes supported by the HSS, and dealing with correctness error. In the following, recall that Boolean layered circuits of width W, depth D and size S are designed so that every layer is computed by applying some gates on inputs only on the previous layer. Our sublinear MPC will can compute such a circuit supporting N with a communication of $O(\omega(1) \cdot S/\log\log S + (D+W) \cdot S^{o(1)} \cdot \mathsf{poly}(N, \lambda))$ for an arbitrarily small tunable $\omega(1)$.

Recipe for Sublinear MPC from HSS from Boyle et al. The intuition why an HSS scheme could be helpful for this task was first brought out by [19] and can be described as follows. Suppose that our HSS scheme supported arbitrary circuits and had no correctness error. Then parties can then run any MPC protocol that distributes shares $\mathsf{sh}_1, \ldots, \mathsf{sh}_N$ that correspond to a homomorphic secret sharing of $\boldsymbol{x}$ of their combined input. The amount of communication per-party for this would be polynomial in the security parameter λ, $|\boldsymbol{x}|$ and the number of parties N. Each party P_ℓ can then locally evaluate on their share sh_ℓ to compute the desired layered circuit C to form evaluation st_ℓ and output this value. For our purposes, let the length of the output be M, which is the width of the last layer. Let $\mathsf{st}_\ell = (\mathsf{st}_{\ell,1}, \ldots, \mathsf{st}_{\ell,M})$. The j^{th} output bit can be reconstructed by adding $\{\mathsf{st}_{\ell,j}\}_{\ell \in [N]}$. This yields an additional communication of $|\mathsf{st}_\ell| = O(M)$

bits per-party. Thus, the total communication is $\mathsf{poly}(\lambda, N, |\boldsymbol{x}|) + O(M)$ which is sublinear in the circuit size.

There are two typical challenges that arise in materializing the intuition above. First, the HSS scheme typically could have an error in output reconstruction. For all currently known schemes with erroneous outputs, leaking out which outputs don't reconstruct correctly can jeopardize security (much as how leaking which LPN samples have error can break the assumption). The second challenge is that typically HSS schemes don't support circuits of arbitrary size; instead, they may only handle circuits of depth $\log S$ or even $\log \log S$. Indeed, our HSS can only handle circuits of depth $c \cdot \log \log S$, for any constant $c < 1$.

To address the challenges of circuit depth, Boyle et al. suggested the following. They suggested dividing the circuit C into $L = S/\log S$ special layers (or $S/\log \log S$ in our case depending on the depth supported by the HSS scheme), such that HSS can be performed from one layer to the next. Unfortunately, this won't work as is because one cannot afford to run a general-purpose MPC for every chunk to generate HSS sharings of the state of the circuit at that layer. This is because the communication for this step could grow as $O(W\,\mathsf{poly}(\lambda, N))$ where W is the width of the circuit. Any savings by running HSS evaluation of circuits with depth $\log S$ (or $\log \log S$ in our case) could be drowned out by multiplicative $\mathsf{poly}(\lambda, N)$ term. To address this, Boyle et al. suggested that for every chunk $i \in [L]$, the MPC is run to generate an HSS sharing of N secret keys $\{\mathsf{sk}_{i,\ell}\}_{i\in[L],\ell\in[N]}$ for a rate-one encryption scheme. Since keys are smaller in size compared to the state of the circuit, this could be done with significantly less communication. The keys $\mathsf{sk}_{i,\ell}$ for every chunk $i \in [L]$ and party P_ℓ is known only to party P_ℓ. The evaluation will follow in encrypt-then-evaluate cycles. Namely, (rate-one) encrypted HSS evaluated shares will be decrypted by HSS, computed upon according the circuit chunk description, and then the resulting HSS evaluations are encrypted by each party using their key for that chunk. This process could go on, but at the end we must reconstruct the output. If our HSS is perfectly/statistically correct each party could simply release the HSS share evaluations unencrypted corresponding to the output layer.

If the HSS evaluations do not satisfy correctness as described above, there could be multiple additional issues. First, the output of computation for each chunk might not be correct. More importantly, for the output layer, when parties reveal the HSS evaluations it could jeopardize security. The fix for the first issue that was proposed was to evaluate the circuit in a fault tolerant fashion using appropriate error correction. Each HSS evaluation will now not only correspond to a decryption followed by evaluation, it will also have an error correction step. To address the second issue, Boyle et al. suggested using MPC at the final layer to reconstruct the final output as opposed to clearly releasing the evaluations. This will introduce additional communication but only about $M \cdot \mathsf{poly}(N, \lambda)$.

Specific Issues in Our Context. We now discuss specific issues that we need to address in our context.

- We can handle circuits of depth $\log \log S$, so we have to implement both error correction and decryption within that depth.

– Each party P_ℓ encrypt their HSS evaluation under their secret key $\mathsf{sk}_{i,\ell}$. Even if the decryption circuit of the encryption is very simple, decrypting $O(N)$ encryptions, followed by HSS reconstruction and evaluation corresponding to the chunk all under the hood of HSS could be too complex for us as such a function has a locality of $\Omega(N)$.

To address error correction issue, we will do naive majority-based error correction. We will have $\kappa = \omega(1)$ copies of HSS shares for the same set of encryption keys, where κ could be any super-constant. Each party will then release κ rate-one encryptions, one for each of the κ HSS evaluations. For the error correction, each HSS evaluation function will simply use majority decoding and compute the majority of κ HSS reconstructions and then apply the circuit corresponding to the chunk. If HSS reconstruction and the decryption are very local, then the whole circuit is very local. This introduces a κ factor larger communication that the previous approach, but we can choose $\kappa = o(\log \log S)$ so that our communication is sublinear.

To implement the encryption with a very local decryption, we rely on sparse LPN yet again. In particular, we revisit the encryption scheme described in Eq. 3, whose decryption circuit has locality equal to the sparsity parameter k. We can handle decryption errors via the same majority-based fault tolerance approach, as described above.

To solve the third issue, we leverage the fact that our encryption scheme is key-homomorphic. Instead of setting up HSS shares for keys $\{\mathsf{sk}_{i,\ell}\}_{\ell\in[N]}$, we set up HSS shares for the sum $\Sigma_{\ell\in[N]}\mathsf{sk}_{i,\ell} = \mathsf{sk}_i$. The ciphertexts encrypting HSS shares under key $\mathsf{sk}_{i,\ell}$ could be homomorphically added to form a ciphertext under sk_i of the HSS reconstruction of the circuit state for the chunk, thanks to the additive reconstruction of our HSS scheme and the additive homomorphism of the encryption scheme. Now, the HSS evaluation could decrypt just the resulting ciphertext encrypted under sk_i as opposed to decrypting N ciphertexts.

While these are the main ideas, there are a number of low-level details that we could not dive into in this overview. The details of our sublinear MPC can be found in Sect. 6.

3 Preliminaries

Notation. Let $\mathbb{N} = \{1, 2, \dots\}$ be the natural numbers, and define $[a, b] := \{a, a+1, \dots, b\}$, $[n] := [1, n]$. Our logarithms are in base 2. For a finite set S, we write $x \leftarrow S$ to denote uniformly sampling x from S. We denote the security parameter by λ; our parameters depend on λ, e.g. $n = n(\lambda)$, and we often drop the explicit dependence. We abbreviate PPT for probabilistic polynomial-time. Our adversaries are non-uniform PPT ensembles $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$. We write $\mathsf{negl}(\lambda)$ to denote negligible functions in λ. Two ensembles of distributions $\{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\{\mathcal{D}'_\lambda\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable if for any non-uniform PPT adversary $\mathcal{A}$ there exists a negligible function negl such that $\mathcal{A}$ can distinguish between the two distributions with probability at most $\mathsf{negl}(\lambda)$.

For $q \in \mathbb{N}$ that is a prime power, we write $\mathbb{F}_q$ to denote the finite field with q elements, and $\mathbb{F}_q^\times$ to denote its non-zero elements. We write vector and matrices in boldcase, e.g. $\boldsymbol{v} \in \mathbb{F}^m$ and $\boldsymbol{A} \in \mathbb{F}^{n \times m}$. We recall the model of arithmetic circuits, and properties of the Bernoulli distribution $\mathsf{Ber}(\mathbb{F}_q, \epsilon)$, in the full version

3.1 Linear Secret Sharing Schemes

We describe linear (multi-)secret sharing schemes, denoted L(M)SS. Looking ahead, our HSS construction will work with an arbitrary LMSS/LSS scheme. The reader can think of the Shamir LMSS as a running example, described in Definition 3.3 below.

Definition 3.1 (Linear Multi-Secret Sharing Scheme). *A N-party, t-private, s-secret* linear multi-secret sharing scheme (LMSS) *over a finite field $\mathbb{F}$ is a tuple of PPT algorithms $\mathsf{LMSS} = (\mathsf{Share}, \mathsf{Rec})$ with the following syntax:*

- $\mathsf{Share}(x_1, \ldots, x_s; \rho) \to (\mathsf{sh}_1, \ldots, \mathsf{sh}_N)$. *Given secrets $x_1, \ldots, x_s \in \mathbb{F}$, this algorithm samples randomness $\rho \in \mathbb{F}^r$ and return shares $\mathsf{sh}_i \in \mathbb{F}^{b_i}$ for all $i \in [N]$. Note that $r, b_1, \ldots, b_N \in \mathbb{N}$ are also part of the description of LMSS. We require $\mathsf{Share} : \mathbb{F}^s \times \mathbb{F}^r \to \mathbb{F}^{b_1} \times \cdots \times \mathbb{F}^{b_N}$ to be a $\mathbb{F}$-linear map.*
- $\mathsf{Rec}(\mathsf{sh}_1, \ldots, \mathsf{sh}_N) \to (x_1, \ldots, x_s)$. *Given shares $(\mathsf{sh}_1, \ldots, \mathsf{sh}_N)$, return the secrets $(x_1, \ldots, x_s)$ or $\perp$. We require $\mathsf{Rec} : \mathbb{F}^{b_1} \times \cdots \times \mathbb{F}^{b_N} \to \mathbb{F}^s$ to be a $\mathbb{F}$-linear map.*

We require the following properties:

- ***Correctness.*** *For any $x_1, \ldots, x_s \in \mathbb{F}$, we have*

$$\Pr_{\rho \in \mathbb{F}^r}\left[\mathsf{Rec}(\mathsf{sh}_1, \ldots, \mathsf{sh}_N) = (x_1, \ldots, x_s) \mid (\mathsf{sh}_1, \ldots, \mathsf{sh}_N) \leftarrow \mathsf{Share}(x_1, \ldots, x_s; \rho)\right] = 1.$$

- ***Privacy.*** *For any tuples $(x_1, \ldots, x_s), (x'_1, \ldots, x'_s) \in \mathbb{F}^s$ and any subset $T \subset [N]$ of size at most t, the following distributions are the same:*

$$\left\{(\mathsf{sh}_i)_{i \in T} \mid (\mathsf{sh}_i)_{i \in [N]} \leftarrow \mathsf{Share}(x_1, \ldots, x_s)\right\} \equiv \left\{(\mathsf{sh}'_i)_{i \in T} \mid (\mathsf{sh}'_i)_{i \in [N]} \leftarrow \mathsf{Share}(x'_1, \ldots, x'_s)\right\}.$$

We define the rate *of LMSS to be $r := s/(b_1 + \cdots + b_N)$. When $s = 1$, we denote the (single-)secret sharing scheme by LSS.*

Notation. We will denote by $[\![x_1 \| \ldots \| x_s]\!]$ a LMSS of s secrets $x_1, \ldots, x_s$, and $[\![x_1 \| \ldots \| x_s]\!]_\ell$ the ℓ'th share for $\ell \in [N]$. When we have a LSS that encodes a single secret, i.e., $s = 1$, its shares are denoted as $[\![x]\!]$ and $[\![x]\!]_\ell$, respectively. When sharing a vector $\boldsymbol{x}$ element-wise using a LSS, we denote the ℓ'th share of $\boldsymbol{x}$ by $[\![\boldsymbol{x}]\!]_\ell$.

Remark 3.2 (LMSS to LSS). A LMSS instance for s secrets can be "split" into s LSS instances $\mathsf{LSS}^{(1)}, \ldots, \mathsf{LSS}^{(s)}$, where for all $\sigma \in [s]$, $\mathsf{LSS}^{(\sigma)}$ shares input x in the σ^{th} slot of LMSS as $(0, \ldots, x, \ldots, 0) = x \cdot \boldsymbol{u}_\sigma$, where $u_\sigma = (0, \cdots, 0, 1, 0, \cdots, 0)$ is the σ'th unit vector of dimension s and with a single 1 at coordinate σ.

These LSS instances can be "merged" back into a LMSS instance in the following sense: there exists an operation Pack, such that for any $\ell \in [N]$, given party P_ℓ's shares of the LSS instances $[\![x_1 \cdot \boldsymbol{u}_1]\!]_\ell^{(1)}, \ldots, [\![x_s \cdot \boldsymbol{u}_s]\!]_\ell^{(s)}$, returns party P_ℓ's share of the LMSS instance:

$$\mathsf{Pack}\left([\![x_1 \cdot \boldsymbol{u}_1]\!]_\ell^{(1)}, \ldots, [\![x_s \cdot \boldsymbol{u}_s]\!]_\ell^{(s)}\right) := \sum_{\sigma \in [s]} [\![x_\sigma \cdot \boldsymbol{u}_\sigma]\!]_\ell^{(\sigma)} = [\![x_1 \| \ldots \| x_s]\!]_\ell \ .$$

We recall the construction of the Shamir LMSS in e.g. [36]. Note that this LMSS achieves the *optimal* tradeoff (see [35]) between the rate and the privacy threshold t, meaning that $r = 1 - t/N$.

Definition 3.3 (Multi-secret Shamir sharing). *Let $\mathbb{F}$ be a finite field, N be the number of parties, and t the privacy threshold. Let $d = \lceil \log_{|\mathbb{F}|}(2N - t) \rceil$, and define $\mathbb{E}$ to be the unique extension field of $\mathbb{F}$ of degree d. Let γ be a primitive element of $\mathbb{E}$ over $\mathbb{F}$. For any $s \leq N - t$, the N-party, t-private, (ds)-secret Shamir LMSS is defined as follows. Pick arbitrary distinct field elements $\alpha_1, \ldots, \alpha_N, \beta_1, \ldots, \beta_s \in \mathbb{E}$.*

- $\mathsf{Share}(x_1, \ldots, x_{ds}) \to (\mathsf{sh}_1, \ldots, \mathsf{sh}_N)$. *On input $(x_1, \ldots, x_{ds}) \in \mathbb{F}^{ds}$, we pack every d elements $(x_{dj}, \ldots, x_{dj+d-1})$ into a field element y_j of $\mathbb{E}$ by setting $y_j = \sum_{i=0}^{d-1} x_{dj+i}\gamma^i$. We then choose a random polynomial $p(X) \in \mathbb{E}[X]$ of degree at most $s+t-1$ such that $p(\beta_j) = y_j$ for all $j \in [s]$. Return $\mathsf{sh}_i = p(\alpha_i)$ for all $i \in [N]$.*
- $\mathsf{Rec}(\mathsf{sh}_1, \ldots, \mathsf{sh}_N) \to (x_1, \ldots, x_{ds})$. *On input the shares $(\mathsf{sh}_1, \ldots, \mathsf{sh}_N)$, we interpolate the unique polynomial $p(X) \in \mathbb{E}[X]$ of degree at most $s + t - 1$ such that $p(\alpha_i) = \mathsf{sh}_i$ for all $i \in I$. We then compute $y_j = p(\beta_j)$ for all $j \in [s]$, and break y_j down into d elements $(x_{dj}, \ldots, x_{dj+d-1})$ of $\mathbb{F}$ (which is a $\mathbb{F}$-linear operation). Return $(x_1, \ldots, x_{ds})$.*

3.2 Homomorphic Secret Sharing

We recall the definition of homomorphic secret sharing schemes from [18,21], in the setting with a single client, an arbitrary number N of servers, and security against t colluding servers. The functions we consider are arithmetic circuits over a finite field.

Definition 3.4. *Let $N(\lambda), t(\lambda), q(\lambda), \epsilon_{\mathsf{corr}}(\lambda)$ be polynomials in λ. A N-party, t-private* homomorphic secret sharing (HSS) *scheme with correctness error ϵ_{corr}, for a class of arithmetic circuits $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ over the finite field $\mathbb{F}_q$, is a tuple of PPT algorithms $\mathsf{HSS} = (\mathsf{Share}, \mathsf{Eval}, \mathsf{Rec})$ with the following syntax:*

- $\mathsf{Share}(\boldsymbol{x}) \to (\mathsf{sh}_1, \ldots, \mathsf{sh}_N)$*: given a vector $\boldsymbol{x} \in \mathbb{F}_q^m$ of field elements, this algorithm returns a secret sharing $(\mathsf{sh}_1, \ldots, \mathsf{sh}_N)$ of $\boldsymbol{x}$.*
- $\mathsf{Eval}(i, f, \mathsf{sh}_i) \to \mathsf{osh}_{f,i}$*: given party index $i \in [N]$, a function $f \in \mathcal{F}_\lambda$ and the share sh_i, this algorithm returns an output share $\mathsf{osh}_{f,i}$.*

– $\mathsf{Rec}(\{\mathsf{osh}_{f,i}\}_{i\in[N]}) \to y_f$: *given the output shares* $\{\mathsf{osh}_{f,i}\}_{i\in[N]}$*, this algorithm returns the final output* y_f *or* $\perp$*.*

We require HSS *to satisfy the following properties:*

– ***Correctness.*** *We say that the* HSS *scheme is* ϵ_{corr}*-correct, if in an honest execution of* HSS *algorithms with error bound* ϵ_{corr}*, one can reconstruct the correct output given the output shares with probability at least* $1 - \epsilon_{\mathsf{corr}}$*. Formally, for all* $\lambda \in \mathbb{N}$*, all functions* $f \in \mathcal{F}_\lambda$*, and inputs* $\boldsymbol{x}$ *to* f*, we have*

$$\Pr\left[\mathsf{Rec}(\{\mathsf{osh}_{f,i}\}_{i\in[N]}) = f(\boldsymbol{x}) \;\middle|\; \begin{array}{l}(\mathsf{sh}_i)_{i\in[N]} \leftarrow \mathsf{HSS.Share}(\boldsymbol{x}) \\ \mathsf{osh}_{f,i} \leftarrow \mathsf{HSS.Eval}(i, f, \mathsf{sh}_i)\ \forall\ i \in [N]\end{array}\right] \geq 1 - \epsilon_{\mathsf{corr}}(\lambda).$$

– ***Security.*** *We say that the* HSS *scheme is secure if any subset of no more than* t *shares of the input* $\boldsymbol{x}$ *reveals no information about* $\boldsymbol{x}$*. Formally, for any sequence of subsets* $\{T_\lambda\}_\lambda$*, where* $T = T_\lambda \subset [N]$ *has size* t*, and any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the advantage of* $\mathcal{A}$ *in the following experiment is bounded by* $1/2 + \mathsf{negl}(\lambda)$ *for a negligible function* negl*.*
  1. $\mathcal{A}$ *picks challenge inputs* $((\boldsymbol{x}_0, \boldsymbol{x}_1), \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, T)$*.*
  2. $\mathcal{C}(\boldsymbol{x}_0, \boldsymbol{x}_1)$ *samples a random bit* $b \leftarrow \{0,1\}$ *and computes* $(\mathsf{sh}_{b,1}, \ldots, \mathsf{sh}_{b,N}) \leftarrow \mathsf{HSS.Share}(\boldsymbol{x}_b)$*.*
  3. $\mathcal{A}$ *outputs a guess* $b' \leftarrow \mathcal{A}_2(\mathsf{st}, (\mathsf{sh}_{b,i})_{i\in T})$*.*

  The advantage of $\mathcal{A}$ *in the above experiment is the probability that* b *equals* b'*.*
– ***Compactness.*** *There exists a polynomial* p *such that for any* $\lambda \in \mathbb{N}$*, any* $i \in [N]$*, any* $f \in \mathcal{F}_\lambda$*, any input* $\boldsymbol{x}$ *to* f*, given* $(\mathsf{sh}_j)_{j\in[N]} \leftarrow \mathsf{Share}(\boldsymbol{x})$*, the output share* $\mathsf{osh}_{f,i} \leftarrow \mathsf{Eval}(i, f, \mathsf{sh}_i)$ *satisfies* $|\mathsf{osh}_{f,i}| \leq p(\lambda)$*. In particular, the output share sizes do not depend on the size of the function* f*.*

Remark 3.5 (Linear Reconstruction). We say that an HSS scheme for a class of arithmetic circuits $\mathcal{F}$ over a field $\mathbb{F}_q$ has *linear reconstruction* if for every $f \in \mathcal{F}$, input $\boldsymbol{x}$ to f, shares $(\mathsf{sh}_j)_{j\in[N]}$, and party $i \in [N]$, the operation $\mathsf{Eval}(i, f, \mathsf{sh}_i) \to \mathsf{osh}_{f,i}$ produces output shares that are vectors of field elements in $\mathbb{F}_q$. Furthermore, the reconstruction $\mathsf{Rec}(\{\mathsf{osh}_{f,i}\}_{i\in[N]}) \to y_f$ consists of applying a $\mathbb{F}_q$-linear map over the output shares $\mathsf{osh}_{f,i}$.

4 Sparse LPN

In this section, we define our *sparse learning parity with noise (*sLPN*)* assumption. sLPN is a natural variant of the LPN assumption, where each column of the public matrix is now k-sparse for a parameter k. First introduced by Alekhnovich [2], who used it for obtaining hardness of approximation results, variants of the sLPN assumption were subsequently used for constructing local pseudorandom generators [8], cryptography with constant computational overhead [42], public-key encryption schemes [4], pseudorandom correlation generators [15] and more. In the full version, we give an overview of known attacks against sLPN that may help establish a plausible concrete tradeoff between the parameters.

Definition 4.1 (Sparse LPN distribution). *Let $\lambda \in \mathbb{N}$ be the security parameter, $n = n(\lambda)$ be the dimension, $m = m(\lambda) \in \mathbb{N}$ the number of samples, $k = k(\lambda) \leq n$ the sparsity parameter, $q = q(\lambda) \in \mathbb{N}$ the field size, and $\epsilon = \epsilon(\lambda) \in (0,1)$ the noise rate. We define the sparse* LPN *distribution $\mathcal{D}_{\mathsf{sLPN},n,m,k,\epsilon,q}$ to be output distribution of the following process:*

- *Sample $\boldsymbol{s} \leftarrow \mathbb{F}_q^{1\times n}$ uniformly at random.*
- *Sample $\boldsymbol{A}$ randomly from $\mathbb{F}_q^{n\times m}$ such that every column of $\boldsymbol{A}$ has exactly k non-zero elements.*
- *Sample $\boldsymbol{e} \leftarrow (\mathsf{Ber}(\mathbb{F}_q, \epsilon))^{1\times m}$, where $\mathsf{Ber}(\mathbb{F}_q, \epsilon)$ returns 0 with probability $1-\epsilon$, and a uniformly random non-zero element of $\mathbb{F}_q$ otherwise.*
- *Compute $\boldsymbol{b} = \boldsymbol{s} \cdot \boldsymbol{A} + \boldsymbol{e}$. Output $(\boldsymbol{A}, \boldsymbol{b})$.*

Similarly, we define $\mathcal{D}_{\mathsf{rand},n,m,k,\epsilon,q}$ to be identical to the distribution $\mathcal{D}_{\mathsf{sLPN},n,m,k,\epsilon,q}$ except that $\boldsymbol{b}$ is chosen uniformly at random from $\mathbb{F}_q^{1\times m}$.

We now state our Sparse LPN assumption. Note the following two parameter choices: $k = \omega(1)$ is a super-constant, and the noise rate $\epsilon = O(n^{-\delta})$ for some $\delta \in (0,1)$.

Assumption 4.1 (The (δ, q)-sLPN Assumption). *Let $\lambda \in \mathbb{N}$ be the security parameter, $\delta \in (0,1)$ be a constant, and $q = q(\lambda)$ is a sequence of prime powers computable in $\mathsf{poly}(\lambda)$ time. We say that the (δ, q)-*sLPN *holds if for all functions $n = n(\lambda), m = m(\lambda), k = k(\lambda), \epsilon = \epsilon(\lambda)$ efficiently computable in $\mathsf{poly}(\lambda)$ time, with $k = \omega(1) \leq n$ and $\epsilon = O(n^{-\delta})$, the following two distributions are computationally indistinguishable:*

$$\{\mathcal{D}_{\mathsf{sLPN},n,m,k,\epsilon,q}\}_{\lambda\in\mathbb{N}} \approx_c \{\mathcal{D}_{\mathsf{rand},n,m,k,\epsilon,q}\}_{\lambda\in\mathbb{N}}.$$

We will also use $\mathsf{sLPN}_{n,m,k,\epsilon,q}$ to refer to the (decisional) sparse LPN problem with fixed parameters, where an adversary needs to distinguish between the two distributions above.

Remark 4.2. In our sparse LPN assumption, we choose $k = \omega(1)$ to be a super-constant (in particular, polylogarithmic) to avoid dealing with syntactical issues arising when k is a constant. Formulations of sparse LPN are well-studied and believed to be hard over $\mathbb{F}_2$ when $k \geq 3$ is a constant (See for example [2,4,34]). Such formulations can support even up to $m = n^{k/2-\epsilon}$ samples for arbitrary constant $\epsilon > 0$. In such cases, however, we require the m columns of $\boldsymbol{A}$ to not admit a sparse combination of columns say $(i_1, \ldots, i_\ell)$ such that $\boldsymbol{A}_{i_i} + \ldots \boldsymbol{A}_{i_\ell} = 0$ for some constant ℓ. This is achieved by requiring the k-regular bipartite graph formed with the columns of $\boldsymbol{A}$ satisfies certain expansion conditions. Unfortunately, this criterion fails to hold for a random graph/random $\boldsymbol{A}$ with inverse polynomial probability $\frac{1}{n^{O(1)}}$.

We could have worked with a stronger assumption, where k is a constant and the matrix $\boldsymbol{A}$ comes from a special distribution of sparse matrices,[9] to achieve a slightly more expressive function class supported by our HSS scheme. Namely, in such a case we could handle $D = O(\log \lambda)$. But for simplicity, we choose to work with super-constant k and uniform k-sparse matrix $\boldsymbol{A}$.

[9] For example, the distribution in the work of Applebaum and Kachlon [9].

4.1 KDM Security

For the security proof of our HSS construction, we will also require the pseudorandomness of a specific secret-dependent sLPN distribution. In this distribution, we essentially encrypt each input x_i and $x_i \cdot s_j$ for all i, j under sLPN. We note that the result proved in this section corresponds to the notion of KDM security with function $f_{x,j}(\boldsymbol{s}) = x \cdot s_j$ for a given index $j \in [n]$ and for all $x \in \mathbb{F}_q$, defined in prior works [5,13,22].

Definition 4.3 (Sparse LPN KDM distribution). *Let $\delta \in (0,1)$ be a constant and $q = q(\lambda)$ a sequence of prime powers efficiently computable in $\mathsf{poly}(\lambda)$ time. Let $\lambda \in \mathbb{N}$ be the security parameter, and $n(\lambda), m(\lambda), k(\lambda), \epsilon(\lambda) \in \mathbb{N}$ be efficiently computable functions of 1^n such that q is a prime power, $k = \omega(1) < n/2$, and $\epsilon = O(n^{-\delta})$. For any sequence of vectors $\{\boldsymbol{x} = \boldsymbol{x}_\lambda\}_\lambda$ where $\boldsymbol{x}_\lambda \in \mathbb{F}_q^m$, we define the distribution $\mathcal{D}^{\mathsf{kdm}}_{\mathsf{sLPN},n,m,k,\epsilon,q}(\boldsymbol{x})$ to be the output of the following process:*

- *Sample $\boldsymbol{s} \leftarrow \mathbb{F}_q^n$.*
- *For all $i \in [m]$, sample a random k-sparse vector $\boldsymbol{a}_i \in \mathbb{F}_q^n$.*
- *For all $i \in [m], j \in [n]$, sample a random $(2k-1)$-sparse vector $\boldsymbol{a}_{i,j} \in \mathbb{F}_q^n$ conditioned on the j^{th} coordinate of $\boldsymbol{a}_{i,j}$ being nonzero.*
- *For every $i \in [m]$, compute $b_i = \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + x_i + e_i$, where $e_i \leftarrow \mathsf{Ber}(\mathbb{F}_q, \epsilon)$.*
- *For every $i \in [m], j \in [n]$, compute $b_{i,j} = \langle \boldsymbol{a}_{i,j}, \boldsymbol{s} \rangle + x_i \cdot s_j + e_{i,j}$, where $e_{i,j} \leftarrow \mathsf{Ber}(\mathbb{F}_q, \epsilon)$.*
- *Output $\{(\mathbf{a}_i, b_i)\}_{i \in [m]}$ and $\{(\mathbf{a}_{i,j}, b_{i,j})\}_{i \in [m], j \in [n]}$.*

Similarly, we define $\mathcal{D}^{\mathsf{kdm}}_{\mathsf{rand},n,m,k,\epsilon,q}$ to be exactly the distribution above except that $b_i, b_{i,j}$ are chosen uniformly at random from $\mathbb{F}_q$ for all $i \in [m]$, $j \in [n]$.

We now show that the above KDM distribution is also computationally indistinguishable from random, assuming the sparse LPN assumption for the same parameters n, k, q, slightly lower noise rate $\epsilon/2$, and a polynomially larger m'. Note that the lemma requires $q > 2$ due to a technical detail, sketched in the technical overview. We give a full proof in the full version, and discuss a few workarounds for the case $q = 2$.

Lemma 4.4 (KDM security of Sparse LPN). *Let $\delta, q, n, m, k, \epsilon$ and $\boldsymbol{x}$ be as specified in Definition 4.3. For $q > 2$ and $k \in \omega(1) \cap o(\sqrt{n})$, assuming the (δ, q)-sLPN assumption holds (c.f. Assumption 4.1), the following distributions are computationally indistinguishable:*

$$\left\{\mathcal{D}^{\mathsf{kdm}}_{\mathsf{sLPN},n,m,k,\epsilon,q}(\boldsymbol{x})\right\}_{n \in \mathbb{N}} \approx_c \left\{\mathcal{D}^{\mathsf{kdm}}_{\mathsf{rand},n,m,k,\epsilon,q}\right\}_{n \in \mathbb{N}}.$$

5 HSS Construction

In this section, we describe our main HSS construction from Sparse LPN (c.f. Assumption 4.1). Our scheme can handle $\log / \log \log$-degree polynomials containing a polynomial number of monomials, and achieve ϵ-correctness for an arbitrary inverse polynomial ϵ.

Function Class. Our HSS supports the function class $\mathcal{P}(\mathbb{F}_q, D, M)$ consists of multivariate polynomials over field $\mathbb{F}_q$ with degree D and number of monomials M. We do not put any constraint on the number of variables m of the polynomial, as long as it is $\mathsf{poly}(\lambda)$. In particular, every function $f \in \mathcal{P}(D, M)$ can be represented as a sum of monomials:

$$f(x_1, \cdots, x_m) = \sum_{\gamma \in [M]} c_\gamma \cdot M_\gamma(x_1, \cdots, x_m) \ ,$$

where $c_\gamma \in \mathbb{F}_q$ is a coefficient and M_γ is a monomial of degree at most D over $\boldsymbol{x}$. Our HSS construction will require polynomials to be represented this way, which is without loss of generality since one can efficiently pre-process any polynomial to be of this form. Looking ahead, our scheme will achieve $D = O\left(\frac{\log \lambda}{\log \log \lambda}\right)$ and $M = \mathsf{poly}(\lambda)$.

In particular, this function class allows us to evaluate arbitrary arithmetic circuits (with fan-in 2) of depth $d = c \cdot \log \log \lambda$, for any $c < 1$. This is because every output of such a circuit can be computed by a degree-2^d polynomial in 2^d number of variables. Since the degree is $D = 2^d = \log^c \lambda = O\left(\frac{\log \lambda}{\log \log \lambda}\right)$, and the number of monomials is $M \leq (2^d)^{2^d} < (\log^c \lambda)^{\log \lambda / \log \log \lambda} = \lambda^c$, we can see that this circuit can be supported by our HSS.

5.1 Scheme Description

Parameters for Sparse LPN. We will use below the Sparse LPN assumption over $\mathbb{F}_q$ with noise rate $n^{-\delta}$ for an arbitrary constant $\delta \in (0, 1)$, chosen such that the assumption holds, and dimension n which is a polynomial in the security parameter λ depending on D and M.

Ingredient: A Linear Secret Sharing Scheme. In our scheme, we require an arbitrary N-party, t-private LSSS scheme $\mathsf{LSS} = (\mathsf{Share}, \mathsf{Rec})$ supported over the field $\mathbb{F}_q$ of computation. For convenience, the reader may think of the Shamir secret sharing scheme (c.f. Definition 3.3). When $N > q$, note that the shares in the Shamir LSSS live in a suitable extension field $\mathbb{E}$ of $\mathbb{F}_q$ such that $|\mathbb{E}| > N$.

Scheme Overview. We now give a high-level overview of our multi-party HSS scheme, expanding on some points made in the technical overview. The full construction is presented in Fig. 2. In our scheme, $\mathsf{HSS.Setup}$ will choose a suitable LSS scheme with reconstruction over $\mathbb{F}_q$ and suitably set sLPN parameters n and k. To share an input $\boldsymbol{x} \in \mathbb{F}_q^m$, $\mathsf{HSS.Share}$ will generate encryptions $\mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x})$, $\mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x} \otimes \boldsymbol{s})$ drawn from the distribution $\mathcal{D}^{\mathsf{kdm}}_{\mathsf{sLPN},n,m,k,\epsilon,q}(\boldsymbol{x})$ in Definition 4.3, along with secret sharings of $\boldsymbol{x}$ and $\boldsymbol{x} \otimes \boldsymbol{s}$. Namely:

- We sample a random k sparse coefficient vector $\boldsymbol{a}_i \in \mathbb{F}_q^n$ for every $i \in [m]$. Similarly, for every $i \in [m], j \in [n]$, we sample a random $2k-1$ sparse vector $\boldsymbol{a}_{i,j} \in \mathbb{F}_q^n$ so that it is non-zero at the j^{th} coordinate.

- To encrypt x_i for $i \in [m]$, we sample a random secret vector $\boldsymbol{s} \leftarrow \mathbb{F}_q^n$ and compute $b_i = \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + x_i + e_i$ for all $i \in [m]$, where $e_i \leftarrow \mathsf{Ber}(\mathbb{F}_q, \epsilon)$ is Bernoulli with noise rate $\epsilon = n^{-\delta}$.
- We also encrypt $x_i \cdot s_j$ for every $i \in [m], j \in [n]$ as follows. We compute $b_{i,j} = \langle \boldsymbol{a}_{i,j}, \boldsymbol{s} \rangle + x_i \cdot s_j + e_{i,j}$, where $e_{i,j} \leftarrow \mathsf{Ber}(\mathbb{F}_q, \epsilon)$ is Bernoulli with noise rate $\epsilon = n^{-\delta}$. Notice that together $\mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x}) := \left\{(\boldsymbol{a}_i, b_i), \{(\boldsymbol{a}_{i,j}, b_{i,j})\}_{j\in[n]}\right\}_{i\in[m]}$ is exactly from the $\mathcal{D}^{\mathsf{kdm}}_{\mathsf{sLPN},n,m,k,\epsilon,q}$ distribution.
- We secret share each x_i, for $i \in [m]$, and products $x_i \cdot s_j$, for $i \in [m]$, $j \in [n]$, using our LSSS scheme. Let us denote the shares of each party P_ℓ with $\ell \in [N]$ by $[\![x_i]\!]_\ell$ and $[\![x_i \cdot s_j]\!]_\ell$, respectively.
- Each party P_ℓ's share $\mathsf{sh}_\ell(\boldsymbol{x})$ consists of $\left(\mathsf{ct}_{\mathbf{s}}(\boldsymbol{x}), \left\{[\![x_i]\!]_\ell, \{[\![x_i \cdot s_j]\!]_\ell\}_{j\in[n]}\right\}_{i\in[m]}\right)$.

Homomorphic Evaluation. To execute $\mathsf{HSS.Eval}$, each party P_ℓ (for $\ell \in [N]$) will perform homomorphic operations on its local shares. Intitially, the parties start with sharings of the form $[\![x_i]\!]_\ell, \left\{[\![x_i \cdot s_j]\!]_\ell\right\}_{j\in[n]}$. Relying additionally on the sparse LPN encodings, we will maintain the invariant that for every intermediate value of computation y, each party P_ℓ stores shares of y and $y \cdot s_j$ for $j \in [n]$. However, as a result of the computation each share can be corrupted by a low-probablity noise over $\mathbb{F}_q$. We will denote these shares using a special notation $\langle\!\langle \alpha \rangle\!\rangle_\ell$ for the intermediate variable α. To be precise $\langle\!\langle \alpha \rangle\!\rangle_\ell = [\![\alpha + e_\alpha]\!]_\ell$ where e_α is a low-probability noise.

The Eval operations will involve both the linear shares and the noisy ciphertext $\mathsf{ct}_{\boldsymbol{x}}(\boldsymbol{s})$, leading to a build-up of noise as each party continue its local computation. The homomorphic operations are performed as follows:

- To add together two intermediate values y and z, each party P_ℓ can just add its local noisy shares:

$$\langle\!\langle y + z \rangle\!\rangle_\ell := \langle\!\langle y \rangle\!\rangle_\ell + \langle\!\langle z \rangle\!\rangle_\ell, \quad \langle\!\langle (y+z) \cdot s_j \rangle\!\rangle_\ell := \langle\!\langle y \cdot s_j \rangle\!\rangle_\ell + \langle\!\langle z \cdot s_j \rangle\!\rangle_\ell \ \forall\, j \in [n].$$

  This operation increases the noise rate only by a factor of 2. Therefore, this extends straightforwardly to handle arbitrary linear combinations of a polynomial number of intermediate values assuming that the initial noise rate is small enough.
- To multiply an intermediate value y with an input x_i, each party P_ℓ will utilize its encryptions $(\boldsymbol{a}_i, b_i), \{(\boldsymbol{a}_{i,j}, b_{i,j})\}_{j\in[n]}$ along with its noisy shares of y to compute:

$$\begin{aligned}
\langle\!\langle x_i \cdot y \rangle\!\rangle_\ell &:= b_i \cdot \langle\!\langle y \rangle\!\rangle_\ell - \sum_{\sigma \in \mathsf{Supp}(\boldsymbol{a}_i)} a_{i,\sigma} \cdot \langle\!\langle y \cdot s_\sigma \rangle\!\rangle_\ell, \\
\langle\!\langle (x_i \cdot y) \cdot s_j \rangle\!\rangle_\ell &:= b_{i,j} \cdot \langle\!\langle y \cdot s_j \rangle\!\rangle_\ell - \sum_{\sigma \in \mathsf{Supp}(\boldsymbol{a}_{i,j})} a_{i,j,\sigma} \cdot \langle\!\langle y \cdot s_\sigma \rangle\!\rangle_\ell \quad \forall\, j \in [n].
\end{aligned} \tag{4}$$

  The reason why the above holds is the following. Recall that $\mathsf{Supp}(\boldsymbol{a})$ denotes the non-zero coordinates of $\boldsymbol{a}$. Without any noise, the above computation

gives the right result, since the equation

$$b_i \approx \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + x_i = \sum_{\sigma \in \mathsf{Supp}(\boldsymbol{a}_i)} a_{i,\sigma} \cdot s_\sigma + x_i,$$

together with the linearity of the shares, imply that

$$b_i \cdot \langle\!\langle y \rangle\!\rangle_\ell - \sum_\sigma a_{i,\sigma} \cdot \langle\!\langle y \cdot s_\sigma \rangle\!\rangle_\ell \approx \langle\!\langle x_i \cdot y \rangle\!\rangle_\ell .$$

In other words, as long as the potentially noisy shares $\langle\!\langle y \rangle\!\rangle_\ell , \{\langle\!\langle y \cdot s_j \rangle\!\rangle_\ell\}_{j \in \mathsf{Supp}(\boldsymbol{a}_i)}$ were noise-free and the sample b_i was also noise free, then the share produced by $\langle\!\langle (x_i \cdot y) \rangle\!\rangle_\ell$ in Eq. 4 will be noise free. A similar argument applies for correctness of computing $\langle\!\langle (x_i \cdot y) \cdot s_j \rangle\!\rangle_\ell$.
The presence of noise affects the correctness as follows. Because both $\boldsymbol{a}_i$ and $\boldsymbol{a}_{i,j}$ have sparsity at most $2k-1$, at every multiplication step the error probability grows by a factor at most $O(k)$. On careful analysis, this growth at each level is just right to set parameters so that we can handle the desired function class.

Using the homomorphic operations described above, we can evaluate any multivariate polynomial $f \in \mathcal{P}(D, M)$, written as $f(x_1, \ldots, x_m) = \sum_{S \in \Lambda} c_S \cdot x_S$, by first locally evaluating each monomial x_S, then locally taking a linear combination of the results. Finally, we describe the reconstruction algorithm $\mathsf{HSS.Rec}$. At the end of the local computations, each party P_ℓ will hold a noisy share $\langle\!\langle y \rangle\!\rangle_\ell$ of the output $y = f(\boldsymbol{x})$. Since these are LSS shares, we may reconstruct a noisy version of y by applying the reconstruction algorithm of LSS.

5.2 Security Analysis

Noise growth analysis. We now analyze the noise growth of our homomorphic operations. Here, we will write explicit noise terms (colored in red) for our noisy shares:

$$\langle\!\langle y \rangle\!\rangle_\ell = [\![y + e_y]\!]_\ell , \quad \langle\!\langle y \cdot s_j \rangle\!\rangle_\ell = [\![y \cdot s_j + e_{y,j}]\!]_\ell \quad \forall\, j \in [n],$$

We start by considering party ℓ's homomorphic multiplication of an input x_i, shared as $\mathsf{sh}_\ell(x_i)$, with an intermediate value y, shared as $\langle\!\langle y \rangle\!\rangle_\ell , \{\langle\!\langle y \cdot s_j \rangle\!\rangle_\ell\}_{\ell \in [N]}$. Recalling that

$$b_i = \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + x_i + e_i = \sum_{\sigma \in \mathsf{Supp}(\boldsymbol{a}_i)} a_{i,\sigma} \cdot s_\sigma + x_i + e_i,$$

HSS from Sparse LPN

Parameters. Number of parties N, threshold t, field $\mathbb{F}_q$, a N-party, (t, t')-private LSSS scheme LSS over $\mathbb{F}_q$, the function class $\mathcal{P}(D, M)$, sLPN parameters (k, n, δ) chosen according to Remark 5.1.

- $\mathsf{Share}(\boldsymbol{x}) \to \{\mathsf{sh}_\ell(\boldsymbol{x})\}_{\ell\in[N]}$. On input $\boldsymbol{x} \in \mathbb{F}_q^m$, sample from the KDM sparse LPN distribution (with secret vector $\boldsymbol{s} \in \mathbb{F}_q^n$)

$$\{\mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x}), \mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x} \otimes \boldsymbol{s})\} := \left\{(\boldsymbol{a}_i, b_i), \{(\boldsymbol{a}_{i,j}, b_{i,j})\}_{j\in[n]}\right\}_{i\in[m]} \leftarrow \mathcal{D}^{\mathsf{kdm}}_{\mathsf{sLPN},n,m,k,\delta,q}(\boldsymbol{x}).$$

  Next, compute secret sharings (for all $i \in [m], j \in [n]$):

$$\mathsf{LSS.Share}(x_i) \to \left\{[\![x_i]\!]_\ell\right\}_{\ell\in[N]}, \quad \mathsf{LSS.Share}(x_i \cdot s_j) \to \left\{[\![x_i \cdot s_j]\!]_\ell\right\}_{\ell\in[N]}.$$

  Return $\mathsf{sh}_\ell(\boldsymbol{x}) := \left(\mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x}), \mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x} \otimes \boldsymbol{s}), \left\{[\![x_i]\!]_\ell, \left\{[\![x_i \cdot s_j]\!]_\ell\right\}_{j\in[n]}\right\}_{i\in[m]}\right)$.
- $\mathsf{Eval}(\ell, P, \mathsf{sh}_\ell(\boldsymbol{x})) \to \mathsf{osh}_{P,\ell}$. Given a party index $\ell \in [N]$, a m-variate polynomial $P \in \mathcal{P}(D, M)$, and the corresponding share $\mathsf{sh}_\ell(\boldsymbol{x})$, we first evaluate each monomial of P, then add these monomials together. The party P_ℓ stores, for each intermediate value z during the computation, a noisy share $\{\!\{z\}\!\}_\ell := \left(\langle\!\langle z\rangle\!\rangle_\ell, \left\{\langle\!\langle z \cdot s_j\rangle\!\rangle_\ell\right\}_{j\in[n]}\right)$ defined as follows:
  - If $z = x_i$ for some $i \in [m]$, set $\langle\!\langle z\rangle\!\rangle_\ell := [\![x_i]\!]_\ell$, $\langle\!\langle z \cdot s_j\rangle\!\rangle_\ell = [\![x_i \cdot s_j]\!]_\ell$ for all $j \in [n]$.
  - If $z = y \cdot x_i$, where y is an intermediate value and x_i is an input, parse $\{\!\{y\}\!\}_\ell$ as above, then compute

$$\langle\!\langle z\rangle\!\rangle_\ell := b_i \cdot \langle\!\langle y\rangle\!\rangle_\ell - \sum_{\sigma\in\mathsf{Supp}(\boldsymbol{a}_i)} a_{i,\sigma} \cdot \langle\!\langle y \cdot s_\sigma\rangle\!\rangle_\ell,$$

$$\langle\!\langle z \cdot s_j\rangle\!\rangle_\ell := b_{i,j} \cdot \langle\!\langle y\rangle\!\rangle_\ell - \sum_{\sigma\in\mathsf{Supp}(\boldsymbol{a}_{i,j})} a_{i,j,\sigma} \cdot \langle\!\langle y \cdot s_\sigma\rangle\!\rangle_\ell \quad \text{for all } j \in [n].$$

  - If $z = \sum_\gamma c_\gamma \cdot y_\gamma$ where c_γ are coefficients and y_γ are intermediate values, parse $\{\!\{y_\gamma\}\!\}_\ell$ as above, then compute

$$\langle\!\langle z\rangle\!\rangle_\ell := \sum_\gamma c_\gamma \cdot \langle\!\langle y_\gamma\rangle\!\rangle_\ell, \quad \langle\!\langle z \cdot s_j\rangle\!\rangle_\ell := \sum_\gamma c_\gamma \cdot \langle\!\langle y_\gamma \cdot s_j\rangle\!\rangle_\ell \quad \text{for all } j \in [n].$$

  Once party P_ℓ has computed the noisy share $\{\!\{z\}\!\}_\ell$ for the final output $P(\boldsymbol{x})$, return $\mathsf{osh}_{P,\ell} := \langle\!\langle z\rangle\!\rangle_\ell$.
- $\mathsf{Rec}(I, \{\mathsf{osh}_\ell\}_{\ell\in I}) \to z$. Given a subset of parties $I \subset [N]$ and corresponding output shares $\{\mathsf{osh}_\ell\}_{\ell\in I}$, return $z \leftarrow \mathsf{LSS.Rec}(I, \{\mathsf{osh}_\ell\}_{\ell\in I})$.

Fig. 2. HSS from Sparse LPN

where we denote the noise term in blue (which has a fixed noise rate $O(n^{-\delta})$), we can compute the following:

$$
\begin{aligned}
& b_i \cdot [\![y + e_y]\!]_\ell - \sum_{\sigma \in \mathsf{Supp}(\boldsymbol{a}_i)} a_{i,\sigma} \cdot [\![y \cdot s_\sigma + e_{y,\sigma}]\!]_\ell \\
&= \left[\!\!\left[\left(\sum_{a_{i,\sigma} \neq 0} a_{i,\sigma} \cdot s_\sigma + x_i + e_i \right) \cdot y + \; b_i \cdot e_y - \left(\sum_{a_{i,\sigma} \neq 0} a_{i,\sigma} \cdot s_\sigma \right) \cdot y - \sum_{a_{i,\sigma} \neq 0} a_{i,\sigma} \cdot e_{y,\sigma} \right]\!\!\right]_\ell \\
&= [\![x_i \cdot y + e_{x_i \cdot y}]\!]_\ell \,, \quad \text{where} \quad e_{x_i \cdot y} = y \cdot e_i + b_i \cdot e_y - \sum_{a_{i,\sigma} \neq 0} a_{i,\sigma} \cdot e_{y,\sigma} .
\end{aligned}
$$

Similarly, we can compute the noise growth for the noisy shares $\langle\!\langle x_i \cdot y \cdot s_j \rangle\!\rangle_\ell$ for all $j \in [n]$, keeping in mind that $b_{i,j} = \sum_{\sigma \in \mathsf{Supp}(\boldsymbol{a}_{i,j})} a_{i,j,\sigma} \cdot s_\sigma + x_i \cdot s_j + e_{i,j}$:

$$
\begin{aligned}
& b_{i,j} \cdot [\![y + e_y]\!]_\ell - \sum_{\sigma \in \mathsf{Supp}(\boldsymbol{a}_{i,j})} a_{i,j,\sigma} \cdot \left[\!\!\left[y \cdot s_j + e_{y,\sigma} \right]\!\!\right]_\ell \\
&= \left[\!\!\left[\left(\sum_{a_{i,j,\sigma} \neq 0} a_{i,j,\sigma} \cdot s_\sigma + x_i \cdot s_j + e_{i,j} \right) \cdot y + b_{i,j} \cdot e_y - \left(\sum_{a_{i,j,\sigma} \neq 0} a_{i,j,\sigma} \cdot s_\sigma \right) \cdot y - \sum_{a_{i,j,\sigma} \neq 0} a_{i,j,\sigma} \cdot e_{y,\sigma} \right]\!\!\right]_\ell \\
&= \left[\!\!\left[x_i \cdot y \cdot s_j + e_{x_i \cdot y, j} \right]\!\!\right]_\ell , \quad \text{where} \quad e_{x_i \cdot y, j} = y \cdot e_{i,j} + b_{i,j} \cdot e_y - \sum_{a_{i,j,\sigma} \neq 0} a_{i,j,\sigma} \cdot e_{y,\sigma} .
\end{aligned}
$$

We observe that after multiplication, the noisy shares $\langle\!\langle x_i \cdot y \rangle\!\rangle$ and $\langle\!\langle x_i \cdot y \cdot s_j \rangle\!\rangle$ both contain one new noise term, and due to the k-sparsity of $\boldsymbol{a}_i$ and $(2k-1)$-sparsity of $\boldsymbol{a}_{i,j}$, aggregate at most $2k$ prior noises. Therefore, the rate of noise increase by a factor of at most $2k+1$. As we started out with noise level $n^{-\delta}$ for some arbitrary $\delta \in (0,1)$, after D steps the noise level is at most $(2k+1)^D n^{-\delta}$.

Next, we consider the noise growth for homomorphic linear combination. Here, the error growth is much less, allowing us to aggregate M error terms for arbitrary $M = \mathsf{poly}(\lambda)$. Namely, for any coefficients $c_\gamma \in \mathbb{F}_q$ and any intermediate noisy shares $\{\!\{y_\gamma\}\!\}_\ell$ with $\gamma \in [M]$, we have:

$$
\begin{aligned}
\sum_{\gamma=1}^{M} c_\gamma \cdot [\![y_\gamma + e_{y_\gamma}]\!]_\ell &= \left[\!\!\left[\sum_{\gamma=1}^{M} c_\gamma \cdot y_\gamma + \sum_{\gamma=1}^{M} c_\gamma \cdot e_{y_\gamma} \right]\!\!\right]_\ell \\
\sum_{\gamma=1}^{M} c_\gamma \cdot \left[\!\!\left[y_\gamma \cdot s_j + e_{y_\gamma, j} \right]\!\!\right]_\ell &= \left[\!\!\left[\sum_{\gamma=1}^{M} c_\gamma \cdot y_\gamma \cdot s_j + \sum_{\gamma=1}^{M} c_\gamma \cdot e_{y_\gamma, j} \right]\!\!\right]_\ell \qquad \forall\, j \in [n].
\end{aligned}
$$

The noise level for the final share grows by a factor of M.

Remark 5.1 (Parameter Selection for HSS). For a given program class $\mathcal{P}(\mathbb{F}_q, D, M)$, given any constant $\delta \in (0,1)$ for which the Sparse LPN assumption holds, and given desired correctness error $\epsilon \in (0,1)$, we need to choose k and n so that

$$
k = \omega(1) \qquad \text{and} \qquad (2k+1)^D \cdot M \cdot n^{-\delta} < \epsilon \; . \tag{5}
$$

When $D = O(\log \lambda / \log\log \lambda)$, $M = \mathsf{poly}(\lambda)$, and $\epsilon = 1/\mathsf{poly}(\lambda)$, we may choose $k = \log^c \lambda$ for a sufficiently small constant $c > 0$, and $n = \lambda^{c'}$ for a sufficiently large constant $c' > 0$ for the above conditions to hold.

Remark 5.2 (Efficiency). Our HSS input share size $|\mathsf{sh}_\ell(\boldsymbol{x})|$, for $\boldsymbol{x} \in \mathbb{F}_q^m$, is equal to $m(n+1)+mn(n+1)+(m+mn)|\mathsf{LSS}| = m(n+1)((n+1)+|\mathsf{LSS}|)$, where $|\mathsf{LSS}|$ is the share size of the linear secret sharing scheme. In particular, by Remark 5.1 this share size depends on program class $\mathcal{P}(D, M)$ supported (since n depends on D and M). In contrast, our HSS output share is just an LSS share. We also note that when LSS is the Shamir secret sharing scheme, the share size $|\mathsf{LSS}|$ is e field elements, where $e \in \mathbb{N}$ is the smallest integer such that $t < q^e$.

For computation, our HSS evaluation only has an overhead of $O(nk|\mathsf{LSS}|)$, since we need to compute on $n+1$ shares $\langle\!\langle y \rangle\!\rangle$, $\{\langle\!\langle y \cdot s_j \rangle\!\rangle\}_{j\in[n]}$, and during multiplication we suffer another $O(k)$ overhead in computing a linear combination of $(k+1)$ terms.

Remark 5.3 (On concrete parameter settings). While Sparse LPN has been extensively studied in the asymptotic setting (see our discussion in Sect. 1.1), there have been little work on determining concrete parameters for the assumption. Prior works such as [6,54] proposed parameters for Sparse LPN in the constant noise regime, while our noise rate is smaller, namely $1/n^\delta$ for an arbitrary $0 < \delta < 1$. Thus, we leave as interesting open questions the task of figuring out concrete parameters for Sparse LPN in our noise regime, and optimizing our HSS to be more efficient.

Remark 5.4 (Constant overhead for constant-degree polynomials). In fact, our HSS construction can be made even more efficient than Remark 5.2 for polynomials of *constant* degree D, where we may achieve $O(k^D \cdot M)$ computation overhead for polynomials with M monomials. If we were to set $k = O(1)$, then the overhead is constant in M. This follows from a more conservative computation of only the necessary values required to evaluate the polynomial. Namely, each homomorphic multiplication of an intermediate value y with an input x_i requires only $(k+1)$ secret shares $\{[\![ys_\sigma]\!]\}_{\sigma\in\mathsf{Supp}(\boldsymbol{a})}$. As a consequence, each degree D monomial computation will touch at most $O(k^D)$ sparse LPN samples and secret shares and involve roughly these many binary additions and multiplications.

From our noise growth analysis above and the subsequent Remark 5.1 on parameter selection, we can conclude the correctness of our HSS construction.

Lemma 5.5 (Correctness of HSS). *Assume the Sparse LPN assumption holds with constant $\delta \in (0,1)$. For any $\epsilon = 1/\mathsf{poly}(\lambda)$, any $D = O(\log\lambda/\log\log\lambda)$, and any $M = \mathsf{poly}(\lambda)$, for parameters n and k chosen according to Remark 5.1, the resulting HSS construction in Fig. 2 is correct except with probability ϵ.*

Remark 5.6 (Decreasing the correctness error). We note that our correctness error can be decreased to $\mathsf{negl}(\lambda)$, at the cost of larger share sizes, more computation, and making reconstruction non-additive. This is done by giving out some $\kappa = \omega(1)$ copies of the same HSS sharings, each with fresh randomness, doing HSS evaluations of the same function for each of these sharings, then taking majority.

We will now show that our HSS scheme is secure. Again, most of the heavy lifting is done in Lemma 4.4, from which our proof follows almost immediately.

Lemma 5.7 (Security of HSS). *Assume the Sparse LPN assumption holds with constant $\delta \in (0,1)$. For any finite field $\mathbb{F}_q$, any number of parties $N \geq 2$, any threshold $t < N$, any $D = O(\log \lambda / \log\log \lambda)$, and any $M = \mathsf{poly}(\lambda)$, with parameters chosen as in Remark 5.1, the N-party HSS construction in Fig. 2 for the class $\mathcal{P}(\mathbb{F}_q, D, M)$ satisfies HSS security with threshold t.*

Proof. Recall that for security of HSS, we need to show that for any subset $I \subset [N]$ of size $|I| \leq t$ and any vectors $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{F}_q^m$, the shares $\{\mathsf{sh}_\ell(\boldsymbol{x})\}_{\ell \in I}$ and $\{\mathsf{sh}_\ell(\boldsymbol{x}')\}_{\ell \in I}$ are computationally indistinguishable. By construction of $\mathsf{HSS.Share}$, these shares consist of two parts, the KDM ciphertexts $\{\mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x}), \mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x} \otimes \boldsymbol{s})\}$ versus $\{\mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x}'), \mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x}' \otimes \boldsymbol{s})\}$, along with the LSS shares $\{[\![\boldsymbol{x}]\!]_\ell, [\![\boldsymbol{x} \otimes \boldsymbol{s}]\!]\}_{\ell \in I}$ versus $\{[\![\boldsymbol{x}']\!]_\ell, [\![\boldsymbol{x}' \otimes \boldsymbol{s}]\!]\}_{\ell \in I}$. The former is indistinguishable due to Lemma 4.4 (for $q = 2$, we may take any of the approaches, detailed in the full version, to conclude Lemma 4.4), and the latter is indistinguishable due to t-privacy of LSS. □

Putting everything together, we get our HSS with desired functionality.

Theorem 5.8 (Multi-party HSS). *Assume the Sparse LPN assumption (c.f. Assumption 4.1) holds. For any number of parties $N \geq 2$, privacy threshold $t < N$, finite field $\mathbb{F}_q$, and error probability $\epsilon = 1/\mathsf{poly}(\lambda)$, there is a N-party, t-private HSS with correctness error ϵ for the function class $\mathcal{P}(\mathbb{F}_q, D, M)$ with degree $D = O(\log \lambda / \log\log \lambda)$ and number of monomials $M = \mathsf{poly}(\lambda)$.*

In the full version, we will present the packed HSS variant that allows us to conclude Theorem 1.3.

6 Sublinear MPC

In this section, we leverage our HSS in Sect. 5 to build a sublinear MPC, with per-party communication dominated by the term $O(S/\log\log S)$, for layered Boolean circuits of size S.[10] Our MPC construction can support an arbitrary $N = \mathsf{poly}(\lambda)$ parties with up to $(N-1)$-out-of-N corruptions.

6.1 Protocol Description

Layered Boolean Circuits. Our MPC construction achieves sublinear communication for the class of *layered Boolean circuits*. A Boolean circuit $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$ is *layered* if its nodes can be partitioned into $D = \mathsf{depth}(\mathsf{C})$ layers $(\mathsf{L}_1, \ldots, \mathsf{L}_D)$ such that any edge (u, v) of C satisfies $u \in L_i$ and $v \in L_{i+1}$ for some $i \leq D-1$. The width $\mathsf{width}(\mathsf{C})$ of a layered circuit C is defined to be

[10] More generally, our construction can generalize to any constant-size field. For simplicity, we only cover the Boolean case.

the maximum number of non-output gates contained in any single layer. In our MPC, we assume that the parties input are $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, concatenated into $\boldsymbol{x} := \boldsymbol{x}_1 \| \boldsymbol{x}_2 \| \ldots \| \boldsymbol{x}_N$. Then $\boldsymbol{x}$ is the overall input to the circuit C, and at the end of the MPC, each party should get $\mathsf{C}(\boldsymbol{x})$.

Remark 6.1 (Circuit Decomposition). From an existing result in [19], for any $d \in \mathbb{N}$, we have a decomposition of C into $L = \lceil D/d \rceil$ *special layers* $(\mathsf{L}_1^\star, \ldots, \mathsf{L}_L^\star)$ such that: (1) two consecutive layers are of distance at most $2d$ from each other, and (2) letting w_i be the width of layer L_i for all $i \in [L]$, we have $\sum_{i=1}^{L} w_i \leq S/d$. We denote by $\mathsf{C}_{i,j}$ the circuit computing the j^{th} output of layer L_{i+1} from the inputs of layer L_i, for all $i \in [L-1], j \in [w_{i+1}]$.

For simplicity, in our MPC construction we will assume that all the inputs to C are in the first layer, and all outputs are in the last layer. This is without loss of generality, as all intermediate values in our construction are represented in the same form; thus, we can "delay" an input until it is needed in an intermediate layer, and similarly delay an output till the end.

Protocol Description. Following the main ideas discussed in Sect. 2.3, we now give our MPC construction in Figs. 3 and 4. In our construction, we assume that each party has access to a broadcast channel. This is simply for ease of presentation, since in the semi-honest model we can simulate such a broadcast channel by letting parties pass messages in a cyclic or star-like fashion. In the full version, we will prove that our MPC is secure, with desired sublinear per-party communication as in Theorem 1.2.

Remark 6.2 (Removing dependence on width). We note that our MPC incurs a communication cost proportional to the circuit width W, due to the use of the public vectors $\{\boldsymbol{a}_{i_2,i_3}\}_{i_2 \in [W], i_3 \in [\kappa]}$. We suggest two main approaches to reduce/eliminate the additive term proportional to the width.

- If the number of parties are large, since this is a semi-honest protocol, each party can be required to output $\frac{W \cdot \kappa}{N}$ such vectors, as opposed to running the MPC for computing $W \cdot \kappa$ such vectors. In this case this additive term can be replaced by a term that grows like $W \cdot \mathsf{poly}(\log N, \log \lambda)/N$. This per party communication becomes sublinear for big enough N.
- Since these vectors are chosen randomly (among all k_{Enc}-sparse vectors), this term can be removed altogether if we are willing to assume any of the following: 1) a uniform random string, 2) a random oracle, or 3) an *explicit* family of k_{Enc}-sparse matrices for which the sparse LPN assumption holds. Any of these assumptions allow the parties to have the description of $\boldsymbol{a}_{i_2,i_3}$ without any further communication.

Sublinear MPC construction, part 1

Local Inputs. Each party P_ℓ, for $\ell \in [N]$, has input $\boldsymbol{x}_\ell$, concatenated into $\boldsymbol{x} = \boldsymbol{x}_1 \| \ldots \| \boldsymbol{x}_\ell$.
Circuit. A layered Boolean circuit $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$ of size S, depth D and width W, decomposed as in Remark 6.1. In particular, we choose depth parameter $d = 0.1 \cdot \log \log S$. This gives special layers $(\mathsf{L}_1^\star, \ldots, \mathsf{L}_L^\star)$, where $L = D/d$, of widths $w_1, \ldots, w_L$ respectively.
Output. The evaluation $\boldsymbol{y} = \mathsf{C}(\boldsymbol{x})$, delivered to each party.
Ingredients.

- Number of repetitions κ, set to be an arbitrary super-constant $\omega(1)$.
- Our encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ from Sparse LPN, described in Equation 3, with noise rate $\epsilon_{\mathsf{Enc}} = 1/(2N\lambda)$. We denote the encryption parameters as follows: the dimension is n_{Enc} and the sparsity is k_{Enc}.
- Our N-party, $(N-1)$-secure HSS in Figure 2, with correctness error $\epsilon_{\mathsf{HSS}} = 1/(2\lambda)$, for the class of Boolean circuits of depth $d' = (0.9 \cdot \log \log S)$ and using the additive secret sharing scheme AdSS. We denote the HSS parameters as follows: the dimension is n_{HSS} and the sparsity is k_{HSS}.
- A general semi-honest MPC protocol, secure against $(N-1)$-out-of-N corruptions, that can evaluate a Boolean circuit C of size S with communication $O(N \cdot S)$ per party. Such a protocol can be based on the existence of oblivious transfer [41].

Protocol Execution:

1. Perform a MPC for the circuit $\mathsf{GenVec} \to \{\boldsymbol{a}_{i_2,i_3}\}_{i_2 \in [W], i_3 \in [\kappa]}$, described as follows:
   For all $i_2 \in [W]$, $i_3 \in [\kappa]$, sample a random k_{Enc}-sparse vector $\boldsymbol{a}_{i_2,i_3} \leftarrow \mathbb{F}_2^{n_{\mathsf{Enc}}}$. Return $\{\boldsymbol{a}_{i_2,i_3}\}_{i_2 \in [W], i_3 \in [\kappa]}$ to all parties.

Fig. 3. Semi-honest sublinear MPC from OTs and Sparse LPN

Sublinear MPC construction, part 2

2. Perform a MPC for the circuit $\mathsf{GenKeyShares} \to \left\{[\![\boldsymbol{k}_{i_1}]\!]_{\ell,i_3}, \mathsf{sh}_{i_1,i_3,\ell}\right\}_{i_1\in[L],i_3\in[\kappa],\ell\in[N]}$, described as follows:
   For all $i_1 \in [0, L-1]$:
   - Sample a random key $\boldsymbol{k}_{i_1} \leftarrow \mathbb{F}_2^{n_{\mathsf{Enc}}}$.
   - For each $i_3 \in [\kappa]$, compute $\{[\![\boldsymbol{k}_{i_1}]\!]_{\ell,i_3}\}_{\ell\in[N]} \leftarrow \mathsf{AdSS.Share}(\boldsymbol{k}_{i_1})$, each time with fresh randomness.
   - For each $i_3 \in [\kappa]$, compute $\{\mathsf{sh}_{i_1,i_3,\ell}\}_{\ell\in[N]} \leftarrow \mathsf{HSS.Share}(\boldsymbol{k}_{i_1})$ for all $i_3 \in [\kappa]$, each time with fresh randomness.

   For each $\ell \in [N]$, return $\{[\![\boldsymbol{k}_{i_1}]\!]_{\ell,i_3}, \mathsf{sh}_{i_1,i_3,\ell}\}_{i_1\in[L],i_3\in[\kappa]}$ to party P_ℓ.
3. For each $\ell \in [N]$, party P_ℓ secret-shares its input $\boldsymbol{x}_\ell$ for κ times $\{[\![\mathsf{st}_{1,i_2,\ell}]\!]_{\ell',i_3}\}_{\ell'\in[N],i_2\in[n],i_3\in[\kappa]} \leftarrow \mathsf{AdSS.Share}(\boldsymbol{x}_\ell)$, and sends $\{[\![\mathsf{st}_{1,i_2,\ell}]\!]_{\ell',i_3}\}_{i_2\in[n],i_3\in[\kappa]}$ to each party $P_{\ell'}$. Then P_ℓ concatenate input shares coming from all other parties to get $\{[\![\mathsf{st}_{1,i_2}]\!]_{\ell,i_3}\}_{i_2\in[n],i_3\in[\kappa]}$.
4. For each layer $i_1 \in [L-1]$ and party index $\ell \in [N]$:
   (a) For each $i_2 \in [w_{i_1}]$, $i_3 \in [\kappa]$, party P_ℓ samples $e_{i_1,i_2,i_3,\ell} \leftarrow \mathsf{Ber}(\mathbb{F}_2, \epsilon_{\mathsf{Enc}})$ and broadcasts its partial ciphertext $\mathsf{ct}_{i_1,i_2,i_3,\ell} = \langle \boldsymbol{a}_{i_2,i_3}, [\![\boldsymbol{k}_{i_1}]\!]_{\ell,i_3}\rangle + [\![\mathsf{st}_{i_1,i_2}]\!]_{\ell,i_3} + e_{i_1,i_2,i_3,\ell}$.
   (b) Party P_ℓ receives partial ciphertexts $\mathsf{ct}_{i_1,i_2,i_3,\ell'}$ from all other parties $P_{\ell'}$ and reconstructs $\mathsf{ct}_{i_1,i_2,i_3} \leftarrow \mathsf{AdSS.Rec}(\{\mathsf{ct}_{i_1,i_2,i_3,\ell}\}_{\ell\in[N]})$.
   (c) For each $i_2' \in [w_{i_1+1}]$, $i_3 \in [\kappa]$, P_ℓ computes $\mathsf{HSS.Eval}(\ell, \mathsf{ComputeLayer}_{i_1,i_2'}, \mathsf{sh}_{i_1,i_3,\ell})$ where $\mathsf{ComputeLayer}_{i_1,i_2'}$ is the following function:
      - Use input $\boldsymbol{k}_{i_1}$ to decrypt $\mathsf{st}_{i_1,i_2,i_3'} \leftarrow \mathsf{Dec}(\boldsymbol{k}_{i_1}, \mathsf{ct}_{i_1,i_2,i_3'})$ for all $i_3' \in [\kappa]$.
      - Compute majority $\mathsf{st}_{i_1,i_2} \leftarrow \mathsf{Majority}(\{\mathsf{st}_{i_1,i_2,i_3'}\}_{i_3'\in[\kappa]})$.
      - Then compute $\mathsf{st}_{i_1+1,i_2'} \leftarrow \mathsf{C}_{i_1,i_2'}(\{\mathsf{st}_{i_1,i_2}\}_{i_2\in[w_{i_1}]})$, where $\mathsf{C}_{i_1,i_2'}$ is the circuit computing the $(i_2')^{th}$ output of the $(i_1+1)^{th}$ layer.
   (d) At this point, each party P_ℓ has shares for the next layer $\{[\![\mathsf{st}_{i_1+1,i_2}]\!]_{\ell,i_3}\}_{i_2\in[w_{i_1+1}],i_3\in[\kappa]}$.
5. Perform a final MPC for the following circuit $\mathsf{FinalRec} \to \boldsymbol{y}$:
   - For each $i_2 \in [m]$, $i_3 \in [\kappa]$, compute $y_{i_2,i_3} \leftarrow \mathsf{AdSS.Rec}(\{[\![\mathsf{st}_{L,i_2}]\!]_{\ell,i_3}\}_{\ell\in[N]})$.
   - For each $i_2 \in [m]$, compute majority $y_{i_2} \leftarrow \mathsf{Majority}(\{\mathsf{osh}_{i_2,i_3}\}_{i_3\in[\kappa]})$.
   - Return $\{y_{i_2}\}_{i_2\in[m]}$ to each party.

Fig. 4. Sublinear MPC construction, continued

Acknowledgments. We thank the anonymous Crypto reviewers for helpful comments and suggestions and the authors of [14] for sharing an early version of their manuscript and answering our questions. Y. Ishai was supported in part by ERC Project NTSC (742754), BSF grant 2018393, and ISF grant 2774/20. A. Jain is supported in part by the Google Research Scholar Award and through various gifts from CYLAB, CMU. H. Lin was supported by NSF grants CNS-1936825 (CAREER), CNS-2026774, a JP Morgan AI Research Award, a Cisco Research Award, and a Simons Collaboration on the Theory of Algorithmic Fairness.

References

1. Abram, D., Damgård, I., Orlandi, C., Scholl, P.: An algebraic framework for silent preprocessing with trustless setup and active security. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 13510, pp. 421–452. Springer (2022). https://doi.org/10.1007/978-3-031-15985-5_15
2. Alekhnovich, M.: More on average case vs approximation complexity. In: 44th FOCS, pp. 298–307. IEEE Computer Society Press (Oct 2003)
3. Allen, S.R., O'Donnell, R., Witmer, D.: How to refute a random CSP. In: Guruswami, V. (ed.) 56th FOCS, pp. 689–708. IEEE Computer Society Press (Oct 2015)
4. Applebaum, B., Barak, B., Wigderson, A.: Public-key cryptography from different assumptions. In: Schulman, L.J. (ed.) 42nd ACM STOC, pp. 171–180. ACM Press (Jun 2010)
5. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_35
6. Applebaum, B., Damgård, I., Ishai, Y., Nielsen, M., Zichron, L.: Secure arithmetic computation with constant computational overhead. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 223–254. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_8
7. Applebaum, B., Ishai, Y., Kushilevitz, E.: On Pseudorandom Generators with Linear Stretch in NC^0. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX/RANDOM -2006. LNCS, vol. 4110, pp. 260–271. Springer, Heidelberg (2006). https://doi.org/10.1007/11830924_25
8. Applebaum, B., Ishai, Y., Kushilevitz, E.: On pseudorandom generators with linear stretch in nc^0. Comput. Complex. **17**(1), 38–69 (2008). https://doi.org/10.1007/s00037-007-0237-6
9. Applebaum, B., Kachlon, E.: Sampling graphs without forbidden subgraphs and unbalanced expanders with negligible error. In: Zuckerman, D. (ed.) 60th FOCS, pp. 171–179. IEEE Computer Society Press (Nov 2019)
10. Applebaum, B., Lovett, S.: Algebraic attacks against random local functions and their countermeasures. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC, pp. 1087–1100. ACM Press (Jun 2016)
11. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_29
12. Boneh, D., et al.: Threshold cryptosystems from threshold fully homomorphic encryption. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 565–596. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_19
13. Boneh, D., Halevi, S., Hamburg, M., Ostrovsky, R.: Circular-Secure encryption from decision Diffie-Hellman. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 108–125. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_7
14. Boyle, E., Coateau, G., Meyer, P.: Sublinear-communication secure multiparty computation does not require FHE. In: Eurocrypt (2023)

15. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018, pp. 896–912. ACM Press (Oct 2018)
16. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 489–518. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_16
17. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Orrù, M.: Homomorphic secret sharing: Optimizations and applications. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 2105–2122. ACM Press (Oct / Nov 2017)
18. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_12
19. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_19
20. Boyle, E., Gilboa, N., Ishai, Y.: Group-Based Secure computation: optimizing rounds, communication, and computation. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 163–193. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_6
21. Boyle, E., Gilboa, N., Ishai, Y., Lin, H., Tessaro, S.: Foundations of homomorphic secret sharing. In: Karlin, A.R. (ed.) ITCS 2018, vol. 94, pp. 21:1–21:21. LIPIcs (Jan 2018)
22. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 3–33. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_1
23. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012, pp. 309–325. ACM (Jan 2012)
24. Brakerski, Z., Lyubashevsky, V., Vaikuntanathan, V., Wichs, D.: Worst-Case hardness for LPN and cryptographic hashing via code smoothing. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 619–635. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_21
25. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) 52nd FOCS, pp. 97–106. IEEE Computer Society Press (Oct 2011)
26. Clear, M., McGoldrick, C.: Multi-identity and multi-key leveled FHE from learning with errors. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 630–656. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_31
27. Couteau, G.: A note on the communication complexity of multiparty computation in the correlated randomness model. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 473–503. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_17
28. Couteau, G.: Personal communication (2023)
29. Couteau, G., Meyer, P.: Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 842–870. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_29

30. Cryan, M., Miltersen, P.B.: On pseudorandom generators in NC^0. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 272–284. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44683-4_24
31. David, B., Dowsley, R., Nascimento, A.C.A.: Universally composable oblivious transfer based on a variant of LPN. In: Gritzalis, D., Kiayias, A., Askoxylakis, I. (eds.) Cryptology and Network Security, pp. 143–158. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-12280-9_10
32. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 93–122. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_4
33. Fazio, N., Gennaro, R., Jafarikhah, T., Skeith, W.E.: Homomorphic secret sharing from Paillier encryption. In: Okamoto, T., Yu, Y., Au, M.H., Li, Y. (eds.) ProvSec 2017. LNCS, vol. 10592, pp. 381–399. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68637-0_23
34. Feige, U.: Relations between average case complexity and approximation complexity. In: 34th ACM STOC, pp. 534–543. ACM Press (May 2002)
35. Fosli, I., Ishai, Y., Kolobov, V.I., Wootters, M.: On the download rate of homomorphic secret sharing. In: Braverman, M. (ed.) 13th Innovations in Theoretical Computer Science Conference (ITCS 2022). Leibniz International Proceedings in Informatics (LIPIcs), vol. 215, pp. 71:1–71:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2022). https://drops.dagstuhl.de/opus/volltexte/2022/15667
36. Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: 24th ACM STOC, pp. 699–710. ACM Press (May 1992)
37. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC, pp. 169–178. ACM Press (May/Jun 2009)
38. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 465–482. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_28
39. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_35
40. Goldreich, O.: Candidate one-way functions based on expander graphs. Electronic Colloquium on Computational Complexity (ECCC) **7**(90) (2000)
41. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC, pp. 218–229. ACM Press (May 1987)
42. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with constant computational overhead. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 433–442. ACM Press (May 2008)
43. Ishai, Y., Lai, R.W.F., Malavolta, G.: A geometric approach to homomorphic secret sharing. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 12711, pp. 92–119. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75248-4_4
44. Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from well-founded assumptions. In: Khuller, S., Williams, V.V. (eds.) STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21–25, 2021, pp. 60–73. ACM (2021)
45. Kothari, P.K., Mori, R., O'Donnell, R., Witmer, D.: Sum of squares lower bounds for refuting any CSP. In: Hatami, H., McKenzie, P., King, V. (eds.) 49th ACM STOC, pp. 132–145. ACM Press (Jun 2017)

46. Lai, R.W.F., Malavolta, G., Schröder, D.: Homomorphic secret sharing for low degree polynomials. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 279–309. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_11
47. Mossel, E., Shpilka, A., Trevisan, L.: On e-biased generators in NC0. In: FOCS, pp. 136–145 (2003)
48. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 735–763. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_26
49. Naor, M., Nissim, K.: Communication preserving protocols for secure function evaluation. In: 33rd ACM STOC, pp. 590–599. ACM Press (Jul 2001)
50. Orlandi, C., Scholl, P., Yakoubov, S.: The Rise of Paillier: homomorphic secret sharing and public-key silent OT. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12696, pp. 678–708. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_24
51. Rivest, R.L., Dertouzos, M.L.: On data banks and privacy homomorphisms (1978)
52. Roy, L., Singh, J.: Large message homomorphic secret sharing from DCR and applications. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12827, pp. 687–717. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84252-9_23
53. Wang, F., Yun, C., Goldwasser, S., Vaikuntanathan, V., Zaharia, M.: Splinter: Practical private queries on public data. In: Akella, A., Howell, J. (eds.) 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27–29, 2017, pp. 299–313. USENIX Association (2017). https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/wang-frank
54. Zichron, L.: Locally computable arithmetic pseudorandom generators. Ph.D. thesis, Master's thesis, School of Electrical Engineering, Tel Aviv University (2017)