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

Quang Dao[*] Yuval Ishai[†] Aayush Jain[‡] Huijia Lin[§]

October 14, 2023

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 or four 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 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.

[*]Carnegie Mellon University. Email: qvd@andrew.cmu.edu.

[†]Technion. Email: yuvali@cs.technion.ac.il.

[‡]Carnegie Mellon University. Email: aayushja@anrew.cmu.edu.

[§]UW. Email: rachel@cs.washington.edu.

Contents

1 Introduction

Homomorphic secret sharing (HSS) [BGI16] is the secret sharing analogue of homomorphic encryption [RD78, Gen09], 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 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 [BGI16, Cou19, CM21], multi-server private information retrieval (PIR) and secure keywords search [GI14, BGI15, WYG$^+$17], generating correlated pseudorandomness [BCGI18, BCG$^+$19], and much more.

The work of Boyle, Gilboa and Ishai [BGI16] 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 [BGI17, BCG$^+$17, BKS19], and diversifying the underlying assumptions to Decision Composite Residuosity (DCR) [FGJS17, OSY21, RS21] or assumptions based on class groups of imaginary quadratic fields [ADOS22]. For more limited function classes, which include constant-degree polynomials, 2-party HSS can be based on different flavors of the Learning Parity with Noise (LPN) assumption [BCG$^+$19, CM21]. 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 [AJL$^+$12, CM15, MW16, DHRW16, BGG$^+$18, BGI$^+$18] or Indistinguishability Obfuscation (iO) [BGI15].[2]

The same "multi-party barrier" exists when it comes to the construction of *sublinear* communication MPC protocols, where 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 [BGI16], 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:

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] 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 [BGI15] 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.

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 [FIKW22]. 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 [Gol00, CM01, Fei02, Ale03, MST03, AOW15, AL16, KMOW17]). The work of [ADI$^+$17] generalized the assumption to large fields $\mathbb{F}_q$. Both of these variants have been used in a number of works (see for example [Ale03, AIK06, IKOS08, ABW10]). The related assumption of local PRGs [Gol00] has also been used in a number of works including the recent construction of program obfuscation scheme [JLS21]. 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 [ABW10] we require δ to be greater than some fixed constant.

The work of [ABW10] 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 [ABW10] 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 Appendix A.

Assumptions	(N, t)	**Function Class**	**Error**
DDH [BGI16, BGI17, BCG+17], DCR [FGJS17]	$(2, 1)$	Branching programs (NC^1)	$1/\mathsf{poly}$
LWE [BKS19]	$(2, 1)$	Branching programs (NC^1)	negl
DCR [OSY21, RS21]	$(2, 1)$	Branching programs (NC^1)	negl
Class Groups [ADOS22]	$(2, 1)$	Branching programs (NC^1)	negl
LPN [BCG+19]	$(2, 1)$	Constant-degree polynomials	none
Quasi-poly LPN [CM21]	$(2, 1)$	Loglog-depth circuits	none
DCR [COS+22]	$(4, \star)$	Constant-degree polynomials	negl
Degree-k Homomorphic Encryption [LMS18, ILM21]	$\left(\lfloor \frac{dt}{k+2} \rfloor, t\right)$	Degree-d polynomials	none [a]
Unconditional (Shamir-based) [FIKW22]	$(dt + 1, t)$	Degree-d polynomials	none
iO and OWF [BGI15]	$(\star, \star)$	Circuits (P/poly)	none
FHE [DHRW16, BGI+18] [b]	$(\star, \star)$	Circuits (P/poly)	negl
Sparse LPN (Ours)	$(\star, \star)$	**Loglog-depth circuits**	$1/\mathsf{poly}$

[a] reconstruction is non-linear
[b] relies on multi-key FHE schemes that can be based on "circular-secure" LWE

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

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 [BGI16], 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 [ABW10]. 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 Figure 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 "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*

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

For a typical circuit that is neither "too deep" nor "too wide", namely $D, W = S^{1-O(1)}$, our MPC protocol is indeed *sublinear* in the circuit size S. In Remark 6.2, we discuss how the dependence on the width W can be removed, assuming either a random oracle or an *explicit* public matrix A for which Sparse LPN holds.

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 $n^{-0.5}$ [Ale03, DDN14] or a specific sparse LPN-type assumption [ABW10].[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 [NN01], 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 $\log^2(n)/n$ [BLVW19].

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 $O((k+1)^d)$ (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 $\mathsf{poly}(\lambda)$ [FGJS17, OSY21, RS21, ADOS22, CM21, BKS19]; the

[3]Namely, the PKE constructed in [ABW10] 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 4.1 for a discussion.

only exception is using FHE [GHS12] 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 [LMS18, ILM21, FIKW22] 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 [FY92], 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 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 [FIKW22], 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 Appendix B.

1.2 Related Work

2-party sublinear MPC. The work of Boyle, Gilboa, and Ishai [BGI16] 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 [FGJS17, OSY21, RS21], poly-modulus LWE [BKS19], or class group assumptions [ADOS22]. More recently, Couteau and Meyer [CM21] 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) [Cou19].

[5]The download rate is the ratio of the output size over the sum of all output share sizes (for details, see [FIKW22])

[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.

Beyond 2 parties. In a very recent and independent work, Boyle, Couteau and Meyer [BCM23] 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 [BCM23]. The results of [BCM23] are based on a novel compiler that obtains a sublinear N-party MPC protocol from an $(N-1)$-party HSS scheme satisfying an extra "Las-Vegas"[7] correctness property, along with a PIR scheme with special properties. (See [BCM23] for details, and Proposition 1 in [BCM23] for a more general framework.) We cannot use the compiler from [BCM23] 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 [BGI16], 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 [BCM23], 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 [BCM23] can leverage HSS schemes with Las-Vegas correctness to yield perfectly correct (3-party or 5-party) MPC protocols [Cou23]. 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 Section A. 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 Section 3.2.

Template from Boyle et. al. Our scheme follows the same high-level template that was first suggested by [BGI16] and has been later adopted in a number of follow-ups such as [BKS19, OSY21, RS21, ADOS22], 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 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. [BGI16, OSY21] 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 [BKS19], 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 [BV11, BGV12] that for some (not so)

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

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 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

$$b_i \llbracket y \rrbracket_\ell - \langle \boldsymbol{a}_i, (\llbracket y s_1 \rrbracket_\ell, \ldots, \llbracket y s_n \rrbracket_\ell)\rangle = \llbracket x_i \cdot y \rrbracket_\ell \quad (1)$$

$$b_{i,j} \llbracket y \rrbracket_\ell - \langle \boldsymbol{a}_{i,j}, (\llbracket y s_1 \rrbracket_\ell, \ldots, \llbracket y s_n \rrbracket_\ell)\rangle = \llbracket x_i \cdot y \cdot s_j \rrbracket_\ell \quad (2)$$

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 $\llbracket x_i \rrbracket_\ell$ and $\llbracket x_i \cdot s_j \rrbracket_\ell$, as one computes shares for degree two computations $x_{i_1} \cdot x_{i_2}$, Equation 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 Equation 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 Equation 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 Section 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 Section 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 Section 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 [BGI16], 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 [BGI16] 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 Equation 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 Section 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 $\boldsymbol{A} \in \mathbb{F}^{n \times m}$.

Arithmetic Circuits. Our description borrows from [Cou19]. Given a finite field $\mathbb{F}$, an *arithmetic circuit* C over $\mathbb{F}$ is a directed acyclic graph of fan-in two gates, where each gate is either addition or multiplication in $\mathbb{F}$. If there is a path between two nodes (v, v') of C, we say that v is an *ancestor* of v', and conversely v' is v's child. The *input nodes* of C, denoted $\mathsf{InpSpace}(\mathsf{C})$, is the set of nodes without any ancestors. Similarly, the *output nodes* of C are the nodes without any children. The size $\mathsf{size}(\mathsf{C})$ of C is the number of nodes, and its depth $\mathsf{depth}(\mathsf{C})$ is the length of the longest path from an input node to an output node.

Bernoulli Distribution. We denote the Bernoulli distribution over a finite field $\mathbb{F}_q$ with noise rate $\epsilon \in (0, 1)$ by $\mathsf{Ber}(\mathbb{F}_q, \epsilon)$; this distribution gives 0 with probability $1 - \epsilon$, and a random non-zero element of $\mathbb{F}_q$ with probability ϵ. We write $e \sim \mathsf{Ber}(\mathbb{F}_q, \epsilon)$ to denote that e comes from the corresponding Bernoulli distribution. We will use the following lemma about the sum of Bernoulli random variables.

Lemma 3.1. *Let $\mathbb{F}_q$ be a finite field, $\epsilon_1, \epsilon_2 \in (0,1)$, and consider $e_1 \leftarrow \mathsf{Ber}(\mathbb{F}_q, \epsilon_1)$, $e_2 \leftarrow \mathsf{Ber}(\mathbb{F}_q, \epsilon_2)$. Then for any $\mu_1, \mu_2 \in \mathbb{F}_q^\times$, we have $\mu_1 e_1 + \mu_2 e_2 \sim \mathsf{Ber}(\mathbb{F}_q, \epsilon')$, where $\epsilon' = \epsilon_1 + \epsilon_2 - \frac{q}{q-1}\epsilon_1\epsilon_2$.*

In particular, when $\epsilon_1 = \epsilon_2 = \epsilon$, we have $\epsilon' = 2\epsilon\left(1 - \frac{q}{2(q-1)}\epsilon\right)$.

Proof. Since a non-zero multiple of a Bernoulli random variable is also Bernoulli (with the same noise rate), we may assume $\mu_1 = \mu_2 = 1$. By a symmetry argument, we can see that $e_1 + e_2$ is also Bernoulli, and thus it remains to show that $\Pr[e_1 + e_2 \neq 0] = \epsilon'$. This is established by cases: $e_1 + e_2 \neq 0$ when exactly one of e_1, e_2 is non-zero, which happens with probability $\epsilon_1(1-\epsilon_2) + \epsilon_2(1-\epsilon_1)$, or when e_1, e_2 are both non-zero but their sum is non-zero, which happens with probability $\frac{q-2}{q-1}\epsilon_1\epsilon_2$. □

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 3.2 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 3.1 (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. [FY92]. Note that this LMSS achieves the *optimal* tradeoff (see [FIKW22]) between the rate and the privacy threshold t, meaning that $r = 1 - t/N$.

Definition 3.2 (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 [BGI15, BGI+18], 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 3.3. *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 3.2 (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 $\mathsf{osh}_{f,i}$.

Definition 3.4 (Upload and download costs of HSS)**.** *Let Π be a N-party, t-private HSS scheme for a function class $\mathcal{F}$. We define the following:*

- *The* upload cost *of Π is*

$$\mathsf{UploadCost}(\Pi) = \max_{f\in\mathcal{F}, \boldsymbol{x}\in\mathsf{InpSpace}(f)} \sum_{i=1}^{N} |\mathsf{sh}_i|\,,$$

  where sh_i is generated as $(\mathsf{sh}_1, \ldots, \mathsf{sh}_N) \leftarrow \Pi.\mathsf{Share}(\boldsymbol{x})$, and $\max$ maximizes over all $f \in \mathcal{F}$, all possible inputs $\boldsymbol{x}$ to f, and the randomness of the $\Pi.\mathsf{Share}$ algorithm.

- *The* download cost *of Π is*

$$\mathsf{DownloadCost}(\Pi) = \max_{f\in\mathcal{F}, \boldsymbol{x}\in\mathsf{InpSpace}(f)} \sum_{i=1}^{N} |\mathsf{osh}_{f,i}|\,,$$

  where is osh_i produced as $\mathsf{osh}_{f,i} \leftarrow \Pi.\mathsf{Eval}(i, f, \mathsf{sh}_i)$, and $\max$ maximizes over all possible $f, \boldsymbol{x}$ and the randomness of the $\Pi.\mathsf{Share}$ and $\Pi.\mathsf{Eval}$ algorithms.

- *The* (download) rate *of* Π *is*

$$\mathsf{Rate}(\Pi) = \min_{f \in \mathcal{F}, \boldsymbol{x} \in \mathsf{InpSpace}(f)} \frac{|y|}{\sum_{i=1}^{N} |\mathsf{osh}_{f,i}|},$$

where y is generated as $y \leftarrow \Pi.\mathsf{Rec}(\{\mathsf{osh}_{f,i}\}_{i \in [N]})$, and max *maximizes over all possible $f, \boldsymbol{x}$, and the randomness of the* $\Pi.\mathsf{Share}$, $\Pi.\mathsf{Eval}$ *and* $\Pi.\mathsf{Rec}$ *algorithms.*[9]

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 [Ale03], who used it for obtaining hardness of approximation results, variants of the sLPN assumption were subsequently used for constructing local pseudorandom generators [AIK08b], cryptography with constant computational overhead [IKOS08], public-key encryption schemes [ABW10], pseudorandom correlation generators [BCGI18] and more. In Appendix A, 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 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 **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.*

[9]If $\Pi.\mathsf{Rec}$ returns $\perp$, we treat $|\perp| = \infty$.

Remark 4.1 (Choice of super-constant k)**.** In our assumption, we choose $k = \omega(1)$ to be a super-constant (in particular, polylogarithmic in our HSS construction) 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 [Fei02, Ale03, ABW10]). 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 $\frac{1}{n^{O(1)}}$.

It is possible to work with a stronger Sparse LPN assumption where $k = O(1)$ is a constant. In this setting, the matrix $\boldsymbol{A}$ must come from a special distribution of sparse matrices with *negligible* probability of a constant-sparse linear relation. Such a distribution, for which Sparse LPN may plausibly hold, may be non-explicitly conjectured to exist [AIK08a], efficiently sampleable over large fields [ADI$^+$17], or over any field [AK19]. This stronger assumption would allow our HSS scheme to achieve a more expressive function class and lower computation overhead; see Remark 5.1 and Remark 5.4 for further discussion.

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 [BHHO08, ACPS09, BKS19].

Definition 4.2 (Sparse LPN KDM distribution)**.** *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. 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 $\boldsymbol{x} = \{\boldsymbol{x}_\lambda\}_{\lambda \in \mathbb{N}}$ 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, and workarounds for the case $q = 2$ are discussed in Remark 4.2.

Lemma 4.1 (KDM security of Sparse LPN). *Let $\delta, q, n, m, k, \epsilon$ and $\boldsymbol{x}$ be as specified in Definition 4.2. 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:*

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

We introduce some notation for the proof. The *support* of a vector $\boldsymbol{a} \in \mathbb{F}_q^n$, denoted $\mathsf{Supp}(\boldsymbol{a}) \subset [n]$, is defined to be the set of non-zero coordinates of $\boldsymbol{a}$. In particular, a k-sparse vector $\boldsymbol{a}$ will have $|\mathsf{Supp}(\boldsymbol{a})| = k$. For a sample $(\boldsymbol{a}, b)$ either from the sLPN or the uniform distribution, define its support to be the same as $\mathsf{Supp}(\boldsymbol{a})$.

Proof. Given an adversary $\mathcal{A}$ distinguishing between the two KDM distributions, we construct an adversary $\mathcal{B}$ distinguishing against the sparse LPN problem $\mathsf{sLPN}_{n,m',k,\epsilon^*,q}$ with the same advantage. We will set ϵ^* so that $\epsilon = 2\epsilon^*\left(1 - \frac{q}{2(q-1)}\epsilon^*\right)$, and set $m' = m \cdot (1 + 2n^4)$, which is a conservative estimate on the number of sparse LPN samples we need.

We start by describing how $\mathcal{B}$ works. $\mathcal{B}$ is assumed to have samples $(\boldsymbol{c}_\ell, d_\ell)_{\ell\in[m']}$ either from the distribution $\mathcal{D}_{\mathsf{sLPN},,n,m',k,\epsilon^*}q$, where $d_\ell = \langle \boldsymbol{c}_\ell, \boldsymbol{s}\rangle + e_\ell$, or from the random distribution $\mathcal{D}_{\mathsf{rand},n,m',k,\epsilon^*,q}$ where d_ℓ is uniformly random. We will use these samples to create a distribution close to the KDM distributions defined above, which will be supplied as input to $\mathcal{A}$. If the underlying samples $(\boldsymbol{c}_\ell, d_\ell)_{\ell\in[m']}$ were generated as per $\mathcal{D}_{\mathsf{sLPN},n,m',k,\epsilon^*,q}$ distribution then the resulting samples will be statistically close to $\mathcal{D}^{\mathsf{kdm}}_{\mathsf{sLPN},n,m,k,\epsilon,q}(\boldsymbol{x})$ distribution, otherwise they will be close to $\mathcal{D}^{\mathsf{kdm}}_{\mathsf{rand},n,m,k,\epsilon,q}$ distribution.

Let the resulting samples be $\{(\boldsymbol{a}_i, b_i)\}_{i\in[m]}$ and $\{(\boldsymbol{a}_{i,j}, b_{i,j})\}_{i\in[m],j\in[n]}$, generated as follows.

1. For each $i \in [m]$, we simply add the i-th input x_i to the i-th sample $(\boldsymbol{c}_i, \boldsymbol{d}_i)$ and output $(\boldsymbol{a}_i, b_i) \leftarrow (\boldsymbol{c}_i, d_i + x_i)$.

2. For each $i \in [m], j \in [n]$, we output the sample $(\boldsymbol{a}_{i,j}, b_{i,j})$ as a carefully chosen linear combination of two k-sparse samples.

   - We examine n^3 disjoint pairs of samples $(\boldsymbol{c}_\ell, d_\ell)$ one by one and continue until we find a pair $(\boldsymbol{c}_{\ell_1}, d_{\ell_1})$ and $(\boldsymbol{c}_{\ell_2}, d_{\ell_2})$ so that the only common index in the support of the coefficient is j, namely, $\mathsf{Supp}(\boldsymbol{c}_{\ell_1}) \cap \mathsf{Supp}(\boldsymbol{c}_{\ell_2}) = \{j\}$. This process will take at most $2n^3$ samples. If we don't succeed in finding such a sample, we abort. Later on, we will show that aborting only happens with negligible probability.
   - Next, we sample a random $r \leftarrow \mathbb{F}_q^\times$, and choose $\mu_1, \mu_2 \in \mathbb{F}_q^\times$ so that $\mu_1 c_{\ell_1,j} + \mu_2 c_{\ell_2,j} = r + x_i$. This can be done because one can pick μ_1 so that $r + x_i - \mu_1 c_{\ell_1,j}$ is non-zero, and then solve for μ_2. This step requires $q > 2$, since for $q = 2$ there is only one choice $\mu_1 = 1$, which may make the above quantity equal to zero.
   - Set $\boldsymbol{a}_{i,j}$ to be equal to $\mu_1\boldsymbol{c}_{\boldsymbol{\ell_1}} + \mu_2\boldsymbol{c}_{\boldsymbol{\ell_2}}$ for all coordinates not equal to j, and the j^{th} coordinate is set to r. Set $b_{i,j}$ to be $\mu_1 d_{\ell_1} + \mu_2 d_{\ell_2}$. In other words, we output $(\boldsymbol{a}_{i,j}, b_{i,j}) = \mu_1 \cdot (\boldsymbol{c}_{\ell_1}, d_{\ell_1}) + \mu_2 \cdot (\boldsymbol{c}_{\ell_2}, d_{\ell_2}) - x_i \cdot \boldsymbol{e}_j$, where $\boldsymbol{e}_j$ is the unit vector at the j^{th} coordinate.

To finish our proof, we have to show two statements. The first is that the above sampling process aborts only with negligible probability. Second, conditioned on non-aborting, the samples generated by the above process are statistically close to the samples in the analogous KDM security distribution. We now argue the first claim.

Claim 4.1. *Let $j \in [n]$. Given $2n^3$ independent random k-sparse vectors grouped into n^3 pairs, with probability $1 - \mathsf{negl}(n)$ there exists a pair $\boldsymbol{c}_{\ell_1}$ and $\boldsymbol{c}_{\ell_2}$ such that $\mathsf{Supp}(\boldsymbol{c}_{\ell_1}) \cap \mathsf{Supp}(\boldsymbol{c}_{\ell_2}) = \{j\}$.*

Proof. Let p be the probability that a pair of randomly sampled k-sparse vectors $(\boldsymbol{c}_{\ell_1}, \boldsymbol{c}_{\ell_2})$ satisfies $\mathsf{Supp}(\boldsymbol{c}_{\ell_1}) \cap \mathsf{Supp}(\boldsymbol{c}_{\ell_2}) = \{j\}$. Then the probability that among n^3 independently sampled such pairs, none of them satisfies the property, is $(1-p)^{n^3}$. We want to show that this probability is $\mathsf{negl}(n)$, which is satisfied if $p = \Omega(k^2/n^2)$ since then $(1-p)^{n^3} \le e^{-p \cdot n^3} \le e^{-O(k^2 n)} = \mathsf{negl}(n)$.

Since $\boldsymbol{c}_{\ell_1}, \boldsymbol{c}_{\ell_2}$ are sampled independently at random, we can think of sampling $\boldsymbol{c}_{\ell_1}$ first so that $j \in \mathsf{Supp}(\boldsymbol{c}_{\ell_1})$, then sampling $\boldsymbol{c}_{\ell_2}$ so that $\mathsf{Supp}(\boldsymbol{c}_{\ell_2}) \cap \mathsf{Supp}(\boldsymbol{c}_{\ell_1}) = \{j\}$. The first event happens with probability $\binom{n-1}{k-1}/\binom{n}{k} = \frac{k}{n}$, and the second happens with probability $\binom{n-k}{k-1}/\binom{n}{k}$. Thus, we have

$$\begin{aligned} p = \frac{k}{n} \cdot \frac{\binom{n-k}{k-1}}{\binom{n}{k}} &= \frac{k^2}{n^2}\left(1 - \frac{k-1}{n-1}\right) \cdots \left(1 - \frac{k-1}{n-k+1}\right) \\ &\ge \frac{k^2}{n^2}\left(1 - \frac{(k-1)^2}{n-k+1}\right) \\ &\ge \frac{k^2}{2n^2}. && \text{(since } k = o(\sqrt{n})\text{)} \end{aligned}$$

□

It remains to show that conditioned on non-abort, $\mathcal{B}$ simulates the right distribution for samples $(\boldsymbol{a}_{i,j}, b_{i,j})$. The j^{th} coordinate of $\boldsymbol{a}_{i,j}$ is non-zero and chosen randomly by construction. Otherwise, $\boldsymbol{a}_{i,j}$ have exactly $2k-2$ non-zero coordinates, coming from $(k-1)$ disjoint coorindates for each of $\mu_1 \boldsymbol{c}_{\ell_1}$ and $\mu_2 \boldsymbol{c}_{\ell_2}$. The position of these non-zero coordinates, along with their values, are both randomly distributed. This is because $(\boldsymbol{c}_{\ell_1}, \boldsymbol{c}_{\ell_2})$, and hence their arbitrary non-zero multiples $(\mu_1 \boldsymbol{c}_{\ell_1}, \mu_2 \boldsymbol{c}_{\ell_2})$, are chosen to be random $(k-1)$-sparse at disjoint positions (aside from the j^{th} coordinate). We now consider the distribution of $b_{i,j}$, which is equal to $\mu_1 d_{\ell_1} + \mu_2 d_{\ell_2}$. When d_{ℓ_1}, d_{ℓ_2} are chosen randomly, so is $b_{i,j}$. When d_{ℓ_1}, d_{ℓ_2} are chosen from the sLPN distribution, so that $d_{\ell_1} = \langle \boldsymbol{c}_{\ell_1}, \boldsymbol{s}\rangle + e_{\ell_1}$ and $d_{\ell_2} = \langle \boldsymbol{c}_{\ell_2}, \boldsymbol{s}\rangle + e_{\ell_2}$ for $e_{\ell_1}, e_{\ell_2} \leftarrow \mathsf{Ber}(\mathbb{F}_q, \epsilon')$, we have that

$$\begin{aligned} b_{i,j} = \mu_1 d_{\ell_1} + \mu_2 d_{\ell_2} &= \langle \mu_1 \boldsymbol{c}_{\ell_1} + \mu_2 \boldsymbol{c}_{\ell_2}, \boldsymbol{s}\rangle + (\mu_1 e_{\ell_1} + \mu_2 e_{\ell_2}) \\ &= \langle \boldsymbol{a}_{i,j}, \boldsymbol{s}\rangle + x_i \cdot s_j + (\mu_1 e_{\ell_1} + \mu_2 e_{\ell_2}). \end{aligned}$$

This finishes the proof since by Lemma 3.1, the sum of two Bernoulli random variables $\mu_1 e_{\ell_1}$, $\mu_2 e_{\ell_2}$ with noise rate ϵ^* is equal to a Bernoulli random variable with noise rate ϵ. □

Remark 4.2. As mentioned in the technical overview and the proof above, our approach fails in the case $\mathbb{F}_2$. Looking ahead, this also affects our HSS construction which can be proved secure under Sparse LPN only for $q > 2$. We see three ways to adapt our result to the case of Boolean circuits:

1. We can embed Boolean computations into computations over $\mathbb{F}_4$, then use our HSS scheme over $\mathbb{F}_4$. This doubles the communication and computation of HSS.

2. We may *conjecture* that Lemma 4.1 holds even for $\mathbb{F}_2$; in other words, there is a way around the technical obstacle mentioned above. This leads to an unchanged HSS construction over $\mathbb{F}_2$.

3. We may assume the security of a *different* sLPN assumption over $\mathbb{F}_2$, where the columns of the matrix $\boldsymbol{A}$ may have either $k-1$ or k entries, each with 50% probability. This change would sidestep the issue in the proof of Lemma 4.1.[10] We also believe that such an assumption,

[10]Essentially, such an assumption allows us to combine either two k-sparse samples whose supports only intersect in the i-th coordinate when $x_i = 0$, or a k-sparse and $(k-1)$-sparse sample with disjoint supports when $x_i = 1$.

which essentially can be viewed as assuming sLPN for sparsity parameters $k-1$ and k at the same time, would have the same security as sLPN.

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. We will begin by describing our main HSS scheme in Section 5.1, analyze its security and correctness in Section 5.2, and finally discuss a packed version of our HSS with optimal download rate in Section 5.3.

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 3.2). 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 Figure 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 4.2, 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 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 $[\![x_i]\!]_\ell, \{[\![x_i \cdot s_j]\!]_\ell\}_{j\in[n]}$. 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 Equation 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 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

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

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

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

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 $i\in[m], j\in[n]$):

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

Return $\mathsf{sh}_\ell(\boldsymbol{x}) := \left(\mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x}), \mathsf{ct}_{\boldsymbol{s}}(\boldsymbol{x}\otimes\boldsymbol{s}), \left\{[\![x_i]\!]_\ell, \{[\![x_i\cdot s_j]\!]_\ell\}_{j\in[n]}\right\}_{i\in[m]}\right)$.

- $\mathsf{Eval}(\ell, P, \mathsf{sh}_\ell(\boldsymbol{x})) \to \mathsf{osh}_{P,\ell}$. 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 $\{\!\{z\}\!\}_\ell := \left(\langle\!\langle z\rangle\!\rangle_\ell, \{\langle\!\langle z\cdot s_j\rangle\!\rangle_\ell\}_{j\in[n]}\right)$ 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 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 $z \leftarrow \mathsf{LSS.Rec}(I, \{\mathsf{osh}_\ell\}_{\ell\in I})$.

Figure 2: HSS from Sparse LPN

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 $b_{i,j} = \sum_{\sigma \in \mathsf{Supp}(\boldsymbol{a}_{i,j})} a_{i,j,\sigma} \cdot s_\sigma + x_i \cdot s_j + e_{i,j}$:

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

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:

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

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.

Alternatively, we may choose $k = O(1)$ (relying on a special matrix distribution as in Remark 4.1), in which case we can further achieve $D = O(\log \lambda)$.

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 Section 1.1), there have been little work on determining concrete parameters for the assumption. Prior works such as [ADI$^+$17, Zic17] 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 5.1 (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 Figure 2 is correct except with probability ϵ.*

Remark 5.5 (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 4.1, from which our proof follows almost immediately.

Lemma 5.2 (Security of HSS)**.** *Assume the (δ, q)-Sparse LPN assumption holds for any constant $\delta \in (0,1)$ and any finite field $\mathbb{F}_q$. Then for 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 Figure 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 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 4.1 (for $q = 2$, we may take any of the approaches, detailed in Remark 4.2, to conclude Lemma 4.1), and the latter is indistinguishable due to t-privacy of LSS. □

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

Packed HSS from Sparse LPN

Ingredients. A N-party, t-private, s-secret LMSS. Let $\mathsf{LSS}^{(1)}, \ldots, \mathsf{LSS}^{(s)}$ be the schemes obtained from LMSS as in Remark 3.1. Let $\mathsf{HSS}^{(1)}, \ldots, \mathsf{HSS}^{(s)}$ be the corresponding N-party, t-private, ϵ_{corr}-correct HSS for $\mathcal{P}(\mathbb{F}_q, D, M)$ from $\mathsf{LSS}^{(1)}, \ldots, \mathsf{LSS}^{(s)}$ respectively, described in Figure 2.

Function class. We support the function class $(\mathcal{P}(\mathbb{F}_q, D, M))^s$ for any $D = O(\log / \log\log)$, $M = \mathsf{poly}(\lambda)$, and correctness error parameter $\epsilon_{\mathsf{corr}} = 1/\mathsf{poly}(\lambda)$.

- $\mathsf{Share}(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(s)}) \to (\mathsf{sh}_1, \ldots, \mathsf{sh}_N)$. Compute $(\mathsf{sh}_1^{(\sigma)}, \ldots, \mathsf{sh}_N^{(\sigma)}) \leftarrow \mathsf{HSS}^{(\sigma)}.\mathsf{Share}(\boldsymbol{x}^{(\sigma)})$ for all $\sigma \in [s]$. Return $\mathsf{sh}_\ell = (\mathsf{sh}_\ell^{(1)}, \ldots, \mathsf{sh}_\ell^{(s)})$ for all $\ell \in [N]$.
- $\mathsf{Eval}(\ell, f, \mathsf{sh}_\ell) \to \mathsf{osh}_\ell$. Parse $f = (f^{(1)}, \ldots, f^{(s)})$. For each $\sigma \in [s]$, compute $\mathsf{HSS}.\mathsf{Eval}(\ell, f^{(\sigma)}, \mathsf{sh}_\ell^{(\sigma)}) \to \mathsf{osh}_\ell^{(\sigma)}$. Return $\mathsf{osh}_\ell = \mathsf{Pack}(\mathsf{osh}_\ell^{(1)}, \ldots, \mathsf{osh}_\ell^{(s)})$.
- $\mathsf{Rec}(I, \{\mathsf{osh}_\ell\}_{\ell \in I}) \to (y^{(1)}, \ldots, y^{(s)})$. Return $\mathsf{LMSS}.\mathsf{Rec}(\{\mathsf{osh}_\ell\}_{\ell \in [N]}) \to (y^{(1)}, \ldots, y^{(s)})$.

Figure 3: Packed HSS variant

Theorem 5.1 (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 $M = \mathsf{poly}(\lambda)$.*

5.3 Packed HSS with improved download rate

We now describe a "packed" version of our HSS construction (given in Figure 3), which allows for lower download cost, and thus higher download rate, by replacing the use of LSS with an arbitrary LMSS. Namely, when we share inputs, we may be able to pack it using e.g. multi-secret Shamir sharing. In particular, assuming LMSS can support s secrets; we can s instances of HSS $f_1, \ldots, f_s$ on inputs $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(s)}$ as follows. First, we will share each input $\boldsymbol{x}^{(\sigma)}$ in the i^{th} slot, i.e. consider the i^{th} LSS obtained from LMSS as in Remark 3.1. We then apply HSS.Eval as before, creating outputs shares of $f_1(x^{(1)}), \ldots, f_s(x^{(s)})$. Finally, each party will first pack all output shares back into a single LMSS share, then reconstruct with LMSS.

Remark 5.6 (Parameter Selection for Packed HSS)**.** In order for our packed HSS construction in Figure 3 to achieve correctness error ϵ_{corr}, we need to set parameters for each $\mathsf{HSS}^{(\sigma)}$, as in Remark 5.1, to give error $\epsilon_{\mathsf{corr}}/s$ (this is to account for Pack adding up s number of output shares).

Theorem 5.2 (Packed HSS)**.** *Let $\mathbb{F}_q$ be an arbitrary finite field, N be the number of parties, and $t < N$ be the corruption threshold. For $s \in \mathbb{N}$ large enough so that there exists a N-party, t-private, s-secret* LMSS *with rate $1 - \frac{t}{N}$* [11]*, Figure 3 gives a N-party, t-private HSS for the function class $(\mathcal{P}(\mathbb{F}_q, D, M))^s$ with the same download rate $1 - \frac{t}{N}$.*

We will show that this download rate is *optimal* for any HSS scheme satisfying one of two conditions, mentioned below.

[11] Such an s exists from Definition 3.2.

Lemma 5.3 (Download rate upper bound)**.** *Let Π be an N-party, t-private, $1/\operatorname{poly}(\lambda)$-correct HSS for a function class $\mathcal{F}$ over $\mathbb{F}$ that contains the* identity *function $f(x_1, \ldots, x_s) = x_1, \ldots, x_s$ for all $x_1, \ldots, x_s \in F$, where $s \in \mathbb{N}$. Assume that Π satisfies one of the following two properties:*

- *Π.*Rec *is linear.*
- *The output shares of Π has size independent of the security parameter.*

Then the download rate of Π is bounded above by $1 - t/N$.

Proof. Let the output share sizes for the identity function be $b_1, \ldots, b_N \in \mathbb{N}$, respectively. This means that the domain and range of Π.Rec is given by $\mathsf{Rec} : \mathbb{F}^{b_1} \times \cdots \times \mathbb{F}^{b_N} \to \mathbb{F}^s$.

We first assume that Π.Rec is linear. In other words, we can represent Rec as a matrix $\boldsymbol{M} \in \mathbb{F}^{s \times B}$, with $B = b_1 + \cdots + b_N$, such that $\mathsf{Rec}(\mathsf{osh}_1, \ldots, \mathsf{osh}_N) = \boldsymbol{M} \cdot (\mathsf{osh}_1, \ldots, \mathsf{osh}_N)^T$. We will show that $\boldsymbol{M}$ has relative distance at least t/N; this implies, together with the singleton bound, that the rate of Π (which is also the rate of $\boldsymbol{M}$) is at most $1 - t/N$. This relative distance comes from the fact that HSS is t-private. In particular, for any corrupted set $T \subset [n]$ with $|T| = t$, the parties in T may not learn anything about any of the reconstructed secrets. This implies that the columns of $\boldsymbol{M}$ corresponding to T (there are $\sum_{\ell \in T} b_\ell$ of them) must be linearly independent, which implies that no vector in the span of $\boldsymbol{M}$ has non-zero coordinates totally contained in the columns corresponding to T. Hence, the relative distance of $\boldsymbol{M}$ is at most $\max_{|T|=t} \left(\sum_{\ell \in T} b_\ell\right) / B \leq t/N$.

We now assume that the latter condition holds. This implies that the output shares must information-theoretically hides the output; we can then invoke the information-theoretic upper bound on the rate of an LMSS in [FIKW22]. $\square$

6 Sublinear MPC

In this section, we leverage our HSS in Section 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.[12] Our MPC construction can support an arbitrary $N = \operatorname{poly}(\lambda)$ 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 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 [BGI16], 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 $i \in [L-1], j \in [w_{i+1}]$.

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

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 Section 2.3, we now give our MPC construction in Figure 4 and 5. 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.

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.

6.2 Security Proofs for Sublinear MPC

In this section, we prove the security and sub-linear communication requirement of our MPC construction, described in Figure 4 and Figure 5. The proof proceeds in three main steps. First, we show that under an honest protocol execution, with suitably chosen parameters, the MPC returns the correct output with probability $1-\mathsf{negl}(\lambda)$. Second, we will show that under the same chosen parameters, our MPC achieve sub-linear per-party communication. Finally, we will establish semi-honest security of our MPC against any coalition of up to $N-1$ corrupt parties. We assume familiarity with definitions of semi-honest security and simulation proofs; for details, see e.g. [Gol04].

We begin by showing that the HSS scheme as required by our MPC construction can indeed compute the circuit $\mathsf{ComputeLayer}_{i_1,i_2}$ for all i_1,i_2.

Lemma 6.1 (HSS supports MPC computation)**.** *For large enough S, the function $\mathsf{ComputeLayer}_{i_1,i_2}$ as described in Figure 5 is computable in depth $0.9\log\log S$ for any $i_1\in[L], i_2\in[w_{i_1}]$.*

Proof. Let $d=0.1\log\log S$. We note that the depth of $\mathsf{ComputeLayer}_{i_1,i_2}$ consists of the following. The first is the depth of decryption, which is 1. Next, we have the depth of the majority function, which is known to be implementable in at most $5d+O(1)$ depth [PPZ92]. Finally, the depth of $\mathsf{C}_{i_1,i_2'}$ is at most $2d$ by our choice of circuit decomposition. Adding everything together gives $7d+O(1)<0.9\log\log S$. □

Remark 6.3 (Parameter Selection for MPC)**.** We detail our choice of parameters here. Looking ahead, these parameters will allow our MPC construction to have $\mathsf{negl}(\lambda)$ correctness error, and also sub-linear per-party communication.

Sublinear MPC construction, part 1

Local Inputs. Each party P_ℓ, for $\ell \in [N]$, has input $\boldsymbol{x}_\ell$, concatenated into $\boldsymbol{x} = \boldsymbol{x}_1 \| \dots \| \boldsymbol{x}_\ell$.
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 $(\mathsf{L}_1^\star, \dots, \mathsf{L}_L^\star)$, where $L = D/d$, of widths $w_1, \dots, w_L$ 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 [GMW87].

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.

Figure 4: Semi-honest sublinear MPC from OTs and Sparse LPN

- We can pick an arbitrary $\kappa = \omega(1)$.
- We need to pick the encryption parameters $n_{\mathsf{Enc}}, k_{\mathsf{Enc}}$ to have noise rate $\epsilon_{\mathsf{Enc}} = 1/(2N\lambda)$. Assuming Sparse LPN (c.f. Assumption 4.1) holds with constant $\delta \in (0,1)$, we need to pick n_{Enc} so that $n_{\mathsf{Enc}}^{-\delta} = \epsilon_{\mathsf{Enc}}$. This translates to $n_{\mathsf{Enc}} = \mathsf{poly}(N, \lambda)$. We can pick k_{Enc} to be arbitrary $\omega(1)$.
- We need to pick the HSS parameters $n_{\mathsf{HSS}}, k_{\mathsf{HSS}}$ to have correctness error $\epsilon_{\mathsf{HSS}} = 1/(2\lambda)$, and to support all Boolean circuits of depth $d' = 0.9 \log\log S$. This depth corresponds to degree at most $2^{d'} = \log^{0.9} S$, and number of monomials at most $2^{2^{d'}} = S^{1/\log^{0.1} S} = S^{o(1)}$. We need to make sure that Equation (5) holds, which in our case is

$$(2k_{\mathsf{HSS}} + 1)^{\log^{0.9} S} \cdot S^{1/\log^{0.1} S} \cdot n_{\mathsf{HSS}}^{-\delta} < \frac{1}{2\lambda}.$$

  Here $\delta \in (0,1)$ is a constant that makes Assumption 4.1 holds. This is equivalent to

$$n_{\mathsf{HSS}} > \left(2\lambda \cdot (2k_{\mathsf{HSS}} + 1)^{\log^{0.9} S} \cdot S^{1/\log^{0.1} S}\right)^{\delta}.$$

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 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.

Figure 5: Sublinear MPC construction, continued

We can thus set $2k_{\mathsf{HSS}} + 1 = \log S$, so that $(2k_{\mathsf{HSS}} + 1)^{\log^{0.9} S} = S^{\log\log S/\log^{0.1} S} = S^{o(1)}$, and thus $n_{\mathsf{HSS}} = \mathsf{poly}(\lambda) \cdot S^{o(1)}$.

Next, we establish the correctness of our MPC. Before doing so, we will prove the following lemma about the correctness of majority decoding on independent noisy data.

Lemma 6.2 (Majority Decoding)**.** *Let λ be the security parameter, and $x \in \{0,1\}$. Consider independent random variables $X_1, \ldots, X_\kappa \in \{0,1\}$ such that for all $i \in [\kappa]$, $X_i = x$ independently with probability at least $1 - 1/\lambda$. Let E be the event that at least $2\kappa/3$ of the X_i's are equal to x. Then for any $\kappa = \omega(1)$, there exists a negligible function* negl *such that E happens with probability $1 - \mathsf{negl}(\lambda)$.*

Proof. For each $i \in [\kappa]$, let $Y_i \in \{0,1\}$ be a random variable such that $Y_i = 1$ if and only if $X_i \neq x$. It follows that $Y_1, \ldots, Y_\kappa$ are independent and $\Pr[Y_i] = 1/\lambda$ for all $i \in [\kappa]$. By tail bounds for the binomial distribution [AG89], the probability that at most $\kappa/3$ of the X_i's are different from x (which is the complement of the event E) is at most $\exp\left(-\kappa \cdot D(1/3 \| 1/\lambda)\right)$, where

$$D(a\|p) := a\log(a/p) + (1-a)\log((1-a)/(1-p))$$

is the *relative entropy* of a with respect to p. We can then calculate $D(1/3\|1/\lambda) := 1/3\log(\lambda/3) + 2/3\log(2/(3(1-1/\lambda))) > 1/3\log(\lambda/3) - O(1)$, which implies $\Pr[E] \geq 1 - \exp(-\kappa(1/3\log(\lambda/3) - O(1))) \geq 1 - O(1/\lambda^{O(\kappa)}) = 1 - \mathsf{negl}(\lambda)$, whenever $\kappa = \omega(1)$. □

Lemma 6.3 (Correctness of MPC)**.** *Assuming all parties faithfully execute the MPC protocol in Figure 4 and 5, for parameters chosen as in Remark 6.3, the protocol returns the correct output with probability $1 - \mathsf{negl}(\lambda)$ over the randomness of all parties.*

Proof. For a layered Boolean circuit C and input $\boldsymbol{x}$, we denote the intermediate values of the circuit at each special layer $\mathsf{L}^\star_{i_1}$ by $\{x_{i_1,i_2}\}_{i_1\in[L],i_2\in[w_{i_1}]}$. In other words, we have $x_{1,i_2} = x_{i_2}$ for all $i_2 \in [w_1]$, and $\mathsf{C}_{i_1,i_2}(x_{i_1,1}, \ldots, x_{i_1,w_{i_1}}) = x_{i_1+1,i_2}$ for all $i_1 \in [L-1], i_2 \in [w_{i_1}]$.

Parties in our MPC will hold *noisy* secret shares of these intermediate values. We denote these noisy intermediate states, each replicated κ times, by $\{\mathsf{st}^\star_{i_1,i_2,i_3}\}_{i_1\in[L],i_2\in[w_{i_1}],i_3\in[\kappa]}$, and their secret share for party P_ℓ by $\left[\!\left[\mathsf{st}^\star_{i_1,i_2}\right]\!\right]_{\ell,i_3}$. We consider these shares at their noisiest moment in the protocol, namely right before computing majority during each HSS evaluation, or during the final MPC.

The intuition for the correctness of our MPC is as follows: in each layer, we will have an error build-up coming from both the noisy encryption and the noisy HSS evaluation. However, if these errors total to at most $1/\lambda$, then by Lemma 6.2, majority decoding will recover the correct value with probability $1 - \mathsf{negl}(\lambda)$. To formalize this intuition, we define the following events for each layer $i_1 \in [L]$:

- For each $i_2 \in [w_{i_1}], i_3 \in [\kappa]$, let $\mathsf{Diff}_{i_1,i_2,i_3}$ be the event that $\mathsf{st}^\star_{i_1,i_2,i_3} \neq x_{i_1,i_2}$.
- For each $i_2 \in [w_{i_1}]$, let E_{i_1,i_2} be the event that $\mathsf{st}^\star_{i_1,i_2,i_3} = x_{i_1,i_2}$ for at least $2\kappa/3$ values of i_3. Let $E_{i_1} = \bigwedge_{i_2\in[w_{i_1}]} E_{i_1,i_2}$.

Note that if E_{i_1} happens for some $i_1 \in [L]$, then majority decoding in layer i_1 will successfully recover x_{i_1,i_2} for all $i_2 \in [w_{i_1}]$. We will show that the events E_{i_1} happen with overwhelming probability.

Claim 6.1. *The events E_{i_1}'s satisfy*

$$\Pr[E_1] \geq 1 - \mathsf{negl}(\lambda), \quad \Pr[E_{i_1} \mid E_{i_1-1} \wedge \cdots \wedge E_1] \geq 1 - \mathsf{negl}(\lambda) \quad \forall\, i_1 \in [2, L].$$

Proof. We will prove that $\Pr[E_{i_1,i_2} \mid E_{i_1-1} \wedge \cdots \wedge E_1] \geq 1 - \mathsf{negl}(\lambda)$ for any $i_1 \in [L]$, $i_2 \in [w_{i_1}]$. By a union bound, this gives $\Pr[E_{i_1} \mid E_{i_1-1} \wedge \cdots \wedge E_1] \geq 1 - w_{i_1} \cdot \mathsf{negl}(\lambda) = 1 - \mathsf{negl}(\lambda)$. By Lemma 6.2 and the fact that the events $\{\mathsf{Diff}_{i_1,i_2,i_3}\}_{i_3 \in [\kappa]}$ are independent (since the HSS evaluation for each $i_3 \in [\kappa]$ uses different HSS shares with fresh randomness), it suffices to show that

$$\Pr[\mathsf{Diff}_{i_1,i_2,i_3} \mid E_{i_1-1} \wedge \cdots \wedge E_1] \leq \frac{1}{\lambda} \quad \forall\, i_1 \in [L], i_2 \in [w_{i_1}], i_3 \in [\kappa]. \tag{6}$$

At the first layer, we start with noise-less secret shares from step 3, and add N independent Bernoulli noise from the partial ciphertexts. This gives a noise rate of at most $N \cdot \epsilon_{\mathsf{Enc}} = 1/(2\lambda) < 1/\lambda$, which establishes Equation (6) for $i_1 = 1$. For each layer $i_1 = 2, \ldots, L-1$, assuming $E_{i_1-1}, \ldots, E_1$ all happen, we have by induction that the values st_{i_1,i_2} after majority decoding are correct for all $i_2 \in [w_{i_1}]$. Thus, this gives the correct share in step 4(d), except for the error probability $\epsilon_{\mathsf{HSS}} = 1/(2\lambda)$ of HSS evaluation. In step 4(a-b) of the next layer, more noise is added as part of the partial ciphertexts; similar to the above, this noise rate is at most $N \cdot \epsilon_{\mathsf{Enc}} = 1/(2\lambda)$. Adding the two errors together establishes Equation (6) for $i_1 = 2, \ldots, L-1$. For $i_1 = L$, we only have the error of the HSS evaluation added before taking majority in the final MPC, which is less than $1/\lambda$. Hence Equation (6) is also valid for $i_1 = L$, finishing the proof.

□

From the claim, we have

$$\begin{aligned}\Pr\left[\bigwedge_{i_1=1}^{L} E_{i_1}\right] &= \prod_{i_1=L}^{2} \Pr[E_{i_1} \mid E_{i_1-1} \wedge \cdots \wedge E_1] \cdot \Pr[E_1] \\ &\geq 1 - L \cdot \mathsf{negl}(\lambda) = 1 - \mathsf{negl}(\lambda).\end{aligned}$$

In other words, with probability $1 - \mathsf{negl}(\lambda)$, the majority decoding is correct at all layers $i_1 \in [L]$, which implies that the output of the MPC is correct. □

We now analyze the per-party communication of our MPC construction and show that, up to lower-order terms, it is indeed sublinear in the circuit size.

Lemma 6.4 (Communication of MPC). *For any $\kappa = \omega(1)$, when parameters are chosen as in Remark 6.3, the N-party MPC protocol in Figure 4 and 5 has per-party communication $O(\kappa \cdot S/\log\log S) + D \cdot S^{o(1)} \cdot \mathsf{poly}(N, \lambda) + W \cdot \mathsf{poly}(\log N, \log \lambda)/N$ when evaluating a layered circuit C of size S, depth D and width W.*

Additionally, assuming any of the assumptions in Remark 6.2, our MPC can be minimally modified to remove the last communication term (depending on W).

Proof. We consider the communication per party of each step in our MPC construction:

- *Step 1.* The communication cost is proportional to the circuit size of GenVec. Moreover, adopting the optimization in Remark 6.2, where each party sends out $W \cdot \kappa/N$ of the sparse vectors $\boldsymbol{a}_{i_2,i_3}$, we may get communication only $W \cdot \mathsf{poly}(\log N, \log \lambda)/N$.
- *Step 2.* The circuit $\mathsf{GenKeyShares}$ in step 2 has size $O(n_{\mathsf{HSS}} \cdot n_{\mathsf{Enc}} \cdot \kappa)$, which gives per-party communication of at most $D \cdot S^{o(1)} \cdot \mathsf{poly}(N, \lambda)$.
- *Step 3.* The per-party communication consists of the party's additive secret sharings, which has total size at most $\kappa \cdot N \cdot n$.

- *Step 4.* For each layer $i_1 \in [L-1]$ and each $i_2 \in [w_{i_1}], i_3 \in [\kappa]$, each party sends a 1-bit ciphertext. Thus the per-party communication is $\sum_{i_1=1}^{L-1} w_{i_1} \cdot \kappa \leq 0.1 \cdot \kappa \cdot S/\log\log S$, by the properties of circuit decomposition (in Remark 6.1).
- *Step 5.* The circuit $\mathsf{FinalRec}$ in this step has size $O(\kappa \cdot m)$, hence the per-party communication is at most $N \cdot m \cdot \mathsf{poly}(\lambda)$.

Adding together these costs finishes our proof. Note that our per-party communication is truly *sub-linear* for large enough S, and for $D = S^{1-O(1)}$ (the layered circuit is not too "tall-and-skinny"), since the polynomial $\mathsf{poly}(N, \lambda)$ can be seen to be *independent* of S. $\square$

Finally, we will prove that our MPC is secure.

Theorem 6.1 (Security of MPC)**.** *Assume the Sparse LPN assumption and the existence of oblivious transfer. Then our MPC construction in Figure 4 and 5 is secure against any corruption pattern in the semi-honest model.*

Proof. Let $\mathcal{C} \subsetneq [N]$ be the set of corrupted parties. We describe a simulator Sim in Figure 6 that outputs a simulated view of the corrupted parties. The proof that this simulated view is indistinguishable from the real view goes through the following sequence of hybrids:

- Hyb_1: this is the view of corrupted parties in the real protocol execution. Aside from their inputs and randomness, the view consists of the following:
  - The MPC messages those parties received in steps 1 and 2 of Figure 4, and in step 5 of Figure 5.
  - The vectors $\{\boldsymbol{a}_{i_2,i_3}\}_{i_2\in[W],i_3\in[\kappa]}$ from the first step.
  - The key shares $\{[\![\boldsymbol{k}_{i_1}]\!]_{\ell,i_3}, \mathsf{sh}_{i_1,i_3,\ell}\}_{i_1\in[L],i_3\in[\kappa],\ell\in\mathcal{C}}$ from the second step.
  - The secret shared inputs $\{[\![\mathsf{st}_{0,i_2}]\!]_{\ell,i_3}\}_{i_2\in[n],i_3\in[\kappa],\ell\in\mathcal{C}}$.
  - For each layer $i_1 \in [L]$, the partial ciphertexts $\{\mathsf{ct}_{i_1,i_2,i_3,\ell}\}_{i_2\in[w_{i_1}],i_3\in[\kappa],\ell\in[N]}$.
- Hyb_2: In this hybrid, we use $\mathcal{S}_{\mathsf{MPC}}$ to simulate the MPC messages in steps 1, 2, and 5. In particular, we have the following *hybrid* simulator Sim_2 that differs from the real protocol execution in the following ways:
  - Similar to the real world, Sim_2 is allowed to see the inputs of all parties.
  - In step 1 of MPC, Sim_2 first generate random k_{Enc}-sparse vectors $\{\boldsymbol{a}_{i_2,i_3}\}_{i_2,i_3}$, then run $\mathcal{S}_{\mathsf{MPC}}$ for GenVec with these vectors as outputs.
  - In step 2 of MPC, Sim_2 first execute $\mathsf{GenKeyShares}$ on its own to get the key shares of $\{\boldsymbol{k}_{i_1}\}_{i_1}$ as outputs, then run $\mathcal{S}_{\mathsf{MPC}}$ for $\mathsf{GenKeyShares}$ with these outputs.
  - In step 5 of MPC, Sim_2 will compute the final shares $\{[\![\mathsf{st}_{L,i_2}]\!]_{\ell,i_3}\}_{i_2,i_3,\ell}$ as in the real world, then run $\mathcal{S}_{\mathsf{MPC}}$ for $\mathsf{FinalRec}$ with these shares as inputs and the circuit evaluation $\mathsf{C}(\boldsymbol{x})$ as output.

  $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_2$: There are two differences compared to Hyb_1. The first is that we swapped out the real messages in steps 1,2,5 of the protocol execution for *simulated* messages, generated by $\mathcal{S}_{\mathsf{MPC}}$. This change is indistinguishable due to the guarantee of $\mathcal{S}_{\mathsf{MPC}}$. The second is that in step 5, we invoke $\mathcal{S}_{\mathsf{MPC}}$ for simulating $\mathsf{FinalRec}$ with the evaluation $\mathsf{C}(\boldsymbol{x})$ as output. By Lemma 6.3, this will be the correct output of $\mathsf{FinalRec}$ with probability $1 - \mathsf{negl}(\lambda)$; hence the second change is also indistinguishable.

Simulator Sim for sublinear MPC

Primitives. Simulator $\mathcal{S}_{\mathsf{MPC}}$ for the generic MPC protocol required in Figure 4.
Inputs. Layered Boolean circuit C, circuit input $\boldsymbol{x} := \boldsymbol{x}_1 \| \ldots \| \boldsymbol{x}_N$, and circuit evaluation $\mathsf{C}(\boldsymbol{x})$.
Output. The simulated view of the corrupted parties $\mathcal{C} \subsetneq [N]$ which consists of their inputs $\{\boldsymbol{x}_i\}_{i\in\mathcal{C}}$ and randomness $\{\boldsymbol{r}_i\}_{i\in\mathcal{C}}$, the evaluation $\mathsf{C}(\boldsymbol{x})$, and simulated messages received from the honest parties.

1. **Initialize the simulated view.** Set $\mathsf{SimView} = \perp$.

2. **Simulate MPC for coefficient vectors.** Sample $\{\boldsymbol{a}_{i_2,i_3}\}_{i_2\in[W],i_3\in[\kappa]}$ from the prescribed distribution generated by GenVec. Then, run simulator $\mathcal{S}_{\mathsf{MPC}}$ for GenVec, which takes no input and returns output $\{\boldsymbol{a}_{i_2,i_3}\}_{i_2\in[W],i_3\in[\kappa]}$. We concatenate $\mathcal{S}_{\mathsf{MPC}}$'s output to $\mathsf{SimView}$.

3. **Simulate MPC for generating key shares.** For each $i_1 \in [L], i_3 \in [\kappa]$, run $\mathsf{AdSS.Share}(\mathbf{0}^{n_{\mathsf{HSS}}}) \rightarrow \{[\![\mathbf{0}]\!]_{\ell,i_3}\}_{\ell\in[N]}$ and $\mathsf{HSS.Share}(\mathbf{0}^{n_{\mathsf{HSS}}}) \rightarrow \{\mathsf{sh}^{\mathbf{0}}_{i_1,i_3,\ell}\}_{\ell\in[N]}$. Then, run simulator $\mathcal{S}_{\mathsf{MPC}}$ for $\mathsf{GenKeyShares}$ with the output for corrupted parties $\{[\![\mathbf{0}]\!]_{\ell,i_3}, \mathsf{sh}^{\mathbf{0}}_{i_1,i_3,\ell}\}_{i_1\in[L],i_3\in[\kappa],\ell\in\mathcal{C}}$. We concatenate $\mathcal{S}_{\mathsf{MPC}}$'s output to $\mathsf{SimView}$.

4. **Simulate sharings of inputs.** For each honest party $\ell \notin \mathcal{C}$, run $\mathsf{AdSS.Share}(\mathbf{0}^{|\boldsymbol{x}_\ell|}) \rightarrow \{[\![\mathsf{st}_{1,i_2,\ell}]\!]_{\ell',i_3}\}_{i_2\in[n],i_3\in[\kappa],\ell'\in[N]}$. Concatenate the shares coming from the honest parties to the corrupted parties $\{[\![\mathsf{st}_{1,i_2,\ell}]\!]_{\ell',i_3}\}_{i_2\in[n],i_3\in[\kappa],\ell\notin\mathcal{C},\ell'\in\mathcal{C}}$ to $\mathsf{SimView}$.

5. **Simulate encryptions in each layer.** For each honest party $\ell \notin \mathcal{C}$, each layer $i_1 \in [L-1]$, and each $i_2 \in [w_{i_1}]$, $i_3 \in [\kappa]$, we simulate $\mathsf{ct}_{i_1,i_2,i_3,\ell}$ by sampling it uniformly from $\{0,1\}$. We concatenate these simulated ciphertexts to $\mathsf{SimView}$.

6. **Simulate final MPC.** Perform computation for the corrupted parties to produce the final shares $\{[\![\mathsf{st}_{L,i_2}]\!]_{\ell,i_3}\}_{\ell\in\mathcal{C},i_3\in[\kappa]}$. Run simulator $\mathcal{S}_{\mathsf{MPC}}$ for $\mathsf{FinalRec}$, which takes these shares as inputs and returns output $\mathsf{C}(\boldsymbol{x})$. We concatenate $\mathcal{S}_{\mathsf{MPC}}$'s output to $\mathsf{SimView}$.

7. **Output** $\mathsf{SimView}$**.**

Figure 6: Simulator for sublinear MPC

- $\{\mathsf{Hyb}_{2i+1}, \mathsf{Hyb}_{2i+2}\}_{i \in [L]}$: In these sequences of hybrids, we will switch out the shares of the key $\boldsymbol{k}_{L+1-i}$ and the partial ciphertexts $\mathsf{ct}_{L+1-i,i_2,i_3,\ell}$ one by one, starting from the bottom layer. In particular, in Hyb_{2i+1}, we have a hybrid simulator Sim_{2i+1} that differs from the previous simulator Sim_{2i} as follows:

  – Instead of sampling the $(L+1-i)^{th}$ key $\boldsymbol{k}_{L+1-i}$ at random, then generate additive and HSS shares of $\boldsymbol{k}_{L+1-i}$, Sim_{2i+1} will generate additive and HSS shares of the all-zero vector $\mathbf{0}^{n_{\mathsf{Enc}}}$. The shares for corrupted parties $\{[\![\mathbf{0}]\!]_{\ell,i_3}, \mathsf{sh}^{\mathbf{0}}_{L+1-i,i_3,\ell}\}_{i_3 \in [\kappa], \ell \in \mathcal{C}}$ will be used for the rest of the MPC computation.

  $\mathsf{Hyb}_{2i} \approx_c \mathsf{Hyb}_{2i+1}$: this follows from the security of HSS and the privacy of additive secret sharing. Namely, the following distributions are computationally indistinguishable:

$$\{[\![\boldsymbol{k}_{L+1-i}]\!]_{\ell,i_3}, \mathsf{sh}_{L+1-i,i_3,\ell}\}_{i_3 \in [\kappa], \ell \in \mathcal{C}} \approx_c \{[\![\mathbf{0}]\!]_{\ell,i_3}, \mathsf{sh}^{\mathbf{0}}_{L+1-i,i_3,\ell}\}_{i_3 \in [\kappa], \ell \in \mathcal{C}}.$$

  This implies that any other MPC computation relying on these shares will give indistinguishable outputs as well.

  Similarly, in Hyb_{2i+2}, we have a hybrid simulator Sim_{2i+2} that is the same as Sim_{2i+1}, except that the partial ciphertexts $\{\mathsf{ct}_{L+1-i,i_2,i_3,\ell}\}_{i_2 \in [w_{L+1-i}], i_3 \in [\kappa], \ell \notin \mathcal{C}}$ coming from honest parties for the $(L+1-i)^{th}$ layer are chosen uniformly from $\{0,1\}$.

  $\mathsf{Hyb}_{2i+1} \approx_c \mathsf{Hyb}_{2i+2}$: this follows from two observations. First, in the previous hybrid we have replaced the key shares for the $(L+1-i)^{th}$ layer, so we can consider the partial ciphertexts as being encrypted under a random key. We then observe that ciphertexts of our sLPN-based encryption scheme under random keys are uniformly random element of $\{0,1\}$.

- Hyb_{2L+3}: In this hybrid, we switch out the sharings of inputs from honest parties to be sharings of the all-zero strings. Namely, the hybrid simulator Sim_{2L+3} in this step follows step 4 of the simulator Sim.

  $\mathsf{Hyb}_{2L+2} \approx_c \mathsf{Hyb}_{2L+3}$: this follows from the privacy of additive secret sharing.

Finally, note that the hybrid simulator Sim_{2L+3} is exactly the same as the simulator Sim. This finishes the proof. □

Acknowledgments. We thank the anonymous Crypto reviewers for helpful comments and suggestions and the authors of [BCM23] 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.

7 References

[ABW10] Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In Leonard J. Schulman, editor, *42nd ACM STOC*, pages 171–180. ACM Press, June 2010.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618. Springer, Heidelberg, August 2009.

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

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO*, volume 13510 of *Lecture Notes in Computer Science*, pages 421–452. Springer, 2022.

[AG89] Richard Arratia and Louis Gordon. Tutorial on large deviations for the binomial distribution. *Bulletin of mathematical biology*, 51(1):125–131, 1989.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in nc^0. In Josep Díaz, Klaus Jansen, José D. P. Rolim, and Uri Zwick, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2006 and 10th International Workshop on Randomization and Computation, RANDOM 2006, Barcelona, Spain, August 28-30 2006, Proceedings*, volume 4110 of *Lecture Notes in Computer Science*, pages 260–271. Springer, 2006.

[AIK08a] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in nc 0. *Computational Complexity*, 17:38–69, 2008.

[AIK08b] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in nc^0. *Comput. Complex.*, 17(1):38–69, 2008.

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

[AK19] Benny Applebaum and Eliran Kachlon. Sampling graphs without forbidden subgraphs and unbalanced expanders with negligible error. In David Zuckerman, editor, *60th FOCS*, pages 171–179. IEEE Computer Society Press, November 2019.

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

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *44th FOCS*, pages 298–307. IEEE Computer Society Press, October 2003.

[AOW15] Sarah R. Allen, Ryan O'Donnell, and David Witmer. How to refute a random CSP. In Venkatesan Guruswami, editor, *56th FOCS*, pages 689–708. IEEE Computer Society Press, October 2015.

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

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

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

[BCM23] Elette Boyle, Geoffroy Coateau, and Pierre Meyer. Sublinear-communication secure multiparty computation does not require FHE. In *Eurocrypt*, 2023.

[BGG$^+$18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 565–596. Springer, 2018.

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

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Heidelberg, August 2016.

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

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

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

[BHHO08] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 108–125. Springer, Heidelberg, August 2008.

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

[BLVW19] Zvika Brakerski, Vadim Lyubashevsky, Vinod Vaikuntanathan, and Daniel Wichs. Worst-case hardness for LPN and cryptographic hashing via code smoothing. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 619–635. Springer, Heidelberg, May 2019.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

[CM01] Mary Cryan and Peter Bro Miltersen. On pseudorandom generators in NC. In Jiri Sgall, Ales Pultr, and Petr Kolman, editors, *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Marianske Lazne, Czech Republic, August 27-31, 2001, Proceedings*, volume 2136 of *Lecture Notes in Computer Science*, pages 272–284. Springer, 2001.

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 630–656. Springer, Heidelberg, August 2015.

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

[COS$^+$22] Ilaria Chillotti, Emmanuela Orsini, Peter Scholl, Nigel P. Smart, and Barry Van Leeuwen. Scooby: Improved multi-party homomorphic secret sharing based on FHE. In Clemente Galdi and Stanislaw Jarecki, editors, *Security and Cryptography for Networks - 13th International Conference, SCN 2022, Amalfi, Italy, September 12-14, 2022, Proceedings*, volume 13409 of *Lecture Notes in Computer Science*, pages 540–563. Springer, 2022.

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

[Cou23] Geoffroy Couteau. Personal communication, 2023.

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 502–534, Virtual Event, August 2021. Springer, Heidelberg.

[DDN14] Bernardo David, Rafael Dowsley, and Anderson C. A. Nascimento. Universally composable oblivious transfer based on a variant of lpn. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis Askoxylakis, editors, *Cryptology and Network Security*, pages 143–158, Cham, 2014. Springer International Publishing.

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

[Fei02] Uriel Feige. Relations between average case complexity and approximation complexity. In *34th ACM STOC*, pages 534–543. ACM Press, May 2002.

[FGJS17] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith III. Homomorphic secret sharing from paillier encryption. In Tatsuaki Okamoto, Yong Yu, Man Ho Au, and Yannan Li, editors, *ProvSec 2017*, volume 10592 of *LNCS*, pages 381–399. Springer, Heidelberg, October 2017.

[FIKW22] Ingerid Fosli, Yuval Ishai, Victor I. Kolobov, and Mary Wootters. On the Download Rate of Homomorphic Secret Sharing. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 71:1–71:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[FKO06] Uriel Feige, Jeong Han Kim, and Eran Ofek. Witnesses for non-satisfiability of dense random 3CNF formulas. In *47th FOCS*, pages 497–508. IEEE Computer Society Press, October 2006.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th ACM STOC*, pages 699–710. ACM Press, May 1992.

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

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 465–482. Springer, Heidelberg, April 2012.

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

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

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(90), 2000.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

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

[ILM21] Yuval Ishai, Russell W. F. Lai, and Giulio Malavolta. A geometric approach to homomorphic secret sharing. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 92–119. Springer, Heidelberg, May 2021.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 60–73. ACM, 2021.

[KMOW17] Pravesh K. Kothari, Ryuhei Mori, Ryan O'Donnell, and David Witmer. Sum of squares lower bounds for refuting any CSP. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th ACM STOC*, pages 132–145. ACM Press, June 2017.

[LMS18] Russell W. F. Lai, Giulio Malavolta, and Dominique Schröder. Homomorphic secret sharing for low degree polynomials. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 279–309. Springer, Heidelberg, December 2018.

[MST03] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On e-biased generators in NC0. In *FOCS*, pages 136–145, 2003.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.

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

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

[PPZ92] Michael S. Paterson, Nicholas Pippenger, and Uri Zwick. Optimal carry save networks. In *Poceedings of the London Mathematical Society Symposium on Boolean Function Complexity*, page 174–201, USA, 1992. Cambridge University Press.

[RD78] Ronald L. Rivest and Michael L. Dertouzos. On data banks and privacy homomorphisms. 1978.

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

[WYG+17] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In Aditya Akella and Jon Howell, editors, *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 299–313. USENIX Association, 2017.

[Zic17] Lior Zichron. *Locally computable arithmetic pseudorandom generators*. PhD thesis, Master's thesis, School of Electrical Engineering, Tel Aviv University, 2017.

A A brief Survey of the Sparse LPN assumption

A.1 Cryptanalysis against Linear Tests

In this section, we provide cryptanalysis of the sparse LPN assumption based on the state-of-the-art attacks, which was beautifully summarized in [CRR21]. In particular, all known attacks on sparse LPN (over an arbitrary finite field $\mathbb{F}$) fall into the *linear test* framework, wherein an adversary distinguishes sparse LPN samples from random by measuring the *bias* of an (adversarially chosen) linear combination of the samples.[13] Under this framework, it can be shown that the best currently known distinguishing attacks for sparse LPN take time at least $\min(2^{\widetilde{O}(d)}, 2^{\widetilde{O}(\eta \cdot n)})$ where d is the dual distance of the coefficient matrix $\mathbf{A}$ (see Definition A.1), n is the dimension of the secret and η is the noise probability.

Thus, when $m = n^{1+(k/2-1)(1-\delta)}$ for some $\delta \in (0,1)$, then the dual distance of the matrix $\mathbf{A}$ with high probability can be shown to be at least $\widetilde{\Omega}(n^{\delta})$ with high probability. See Lemma A.1 for the precise statement.

Definition A.1 (Dual Distance)**.** *Let $\mathbb{F}$ be a finite field and $n < m$ be natural numbers. The* dual distance *of a matrix $\boldsymbol{A} \in \mathbb{F}^{n \times m}$, denoted $\mathsf{dd}(\boldsymbol{A})$, is defined to be the distance of the* dual *code $\boldsymbol{H} \in \mathbb{F}^{(m-n)\times m}$ (such that $\boldsymbol{A}^T \cdot \boldsymbol{H} = \mathbf{0}$).*

In the following, we denote by $\mathsf{SpMat}(n, m, k)$ the set of $n \times m$ matrices where each column is exactly k-sparse (over some finite field $\mathbb{F}$ which is implied by the context).

Lemma A.1 (Most sparse matrices have large dual distance)**.** *For a fixed finite field $\mathbb{F}$, given any $k = k(n) \geq 3, m = m(n) \geq n$, there exist a constant $\alpha > 0$,[14] such that for large enough n, letting $t = \left(\frac{m}{n}\right)^{\frac{1}{k/2-1}}$, we have*

$$\Pr\left[\mathsf{dd}(\boldsymbol{A}) \leq \frac{1}{e} \cdot \frac{n}{kt} \middle| \boldsymbol{A} \leftarrow \mathsf{SpMat}(n, m, k)\right] < \alpha \left(\frac{2kt}{n}\right)^{k-2}.$$

Note that Lemma A.1 is a generalization of [CRR21, Theorem 7], handling the case of arbitrary m instead of $m = c \cdot n$ for some constant $c > 0$. From this lemma, we can deduce that sparse LPN is secure in our parameter regime of $k \in (\omega(1), O(\log n))$, $m = \mathsf{poly}(n)$, and $\eta = 1/n^{\delta}$ for any constant $\delta \in (0,1)$, since the right-hand side of the above inequality becomes $\mathsf{negl}(n)$, and conditioned on this negligible probability not happening, any linear test takes time at least $2^{\widetilde{O}(\eta \cdot n)}$ which is sub-exponential in n.

We now provide a proof of Lemma A.1 similar to the proof in [CRR21].

Proof. We begin with some definitions. Given a matrix $\boldsymbol{A} \in \mathbb{F}^{n \times m}$ and a subset $S \subseteq [m]$ of the columns of A, we define $\mathsf{N}_A(S) = \{j \in [n] \mid \exists i \in S \text{ such that } A_{i,j} \neq 0\}$ to be the set of *row*

[13] For a precise statement, see [CRR21, Definition 5].

[14] A crude upper bound from our proof gives $\alpha < 335$.

neighbors of S in A. We say that $\boldsymbol{A}$ is (d,α)-expanding if for every subset $S \subseteq [m]$ of the rows of A with $|S| \le d$, it holds that $|\mathsf{N}_A(S)| > \alpha \cdot |S|$. By an existing argument in [CRR21], we know that $\mathsf{dd}(\boldsymbol{A}) \le d$ implies $\boldsymbol{A}$ is *not* $(d, k/2)$-expanding. Thus, it suffices to upper bound the latter probability for a random column k-sparse matrix $\boldsymbol{A}$.

Consider a random column k-sparse matrix $\boldsymbol{A} \in \mathbb{F}^{n\times m}$, where $m = C \cdot n^c$ for constants $C, c > 0$. Given a s-size subset $S \subset [m]$ and a $ks/2$-size subset $T \subset [n]$, the probability that the neighbors of S in $\boldsymbol{A}$ are contained inside T is at most $\left(\frac{ks}{2n}\right)^{ks}$. Applying a union bound over all sets S and T, with $|S| = s \in [2, d]$, we have:

$$\begin{aligned}
\Pr[\boldsymbol{A} \text{ is not } (d,k/2)\text{-expanding}] &\le \sum_{s=2}^{d} \binom{m}{s} \cdot \binom{n}{ks/2} \cdot \left(\frac{ks}{2n}\right)^{ks} \\
&\le \sum_{s=2}^{d} \left(\frac{em}{s}\right)^{s} \cdot \left(\frac{en}{ks/2}\right)^{ks/2} \cdot \left(\frac{ks}{2n}\right)^{ks} \\
&= \sum_{s=2}^{d} \left(e^{k/2+1} \cdot \left(\frac{k}{2}\right)^{k/2} \cdot \frac{m}{n} \cdot \frac{1}{n^{k/2-1}} \cdot s^{k/2-1}\right)^{s} \\
&= \sum_{s=2}^{d} \left(e^{\frac{k/2+1}{k/2-1}} \cdot \left(\frac{k}{2}\right)^{\frac{k/2}{k/2-1}} \cdot \frac{t}{n} \cdot s\right)^{s(k/2-1)} \\
&< 167 \cdot \sum_{s=2}^{d} \left(\frac{kt}{n} \cdot s\right)^{s(k/2-1)} .
\end{aligned}$$

In the above, the second inequality is due to the fact that $\binom{a}{b} \le \left(\frac{ea}{b}\right)^{b}$, and the last inequality is due to the calculation

$$e^{\frac{k/2+1}{k/2-1}} \cdot \left(\frac{k}{2}\right)^{\frac{k/2}{k/2-1}} \le e^{5} \cdot \left(\frac{3}{2}\right)^{\frac{1}{3/2-1}} \cdot \frac{k}{2} < 167 \cdot k \qquad \text{for all } k \ge 3.$$

Let $N = \frac{n}{kt}$, and consider the function $f(s) = (s/N)^{s(k/2-1)}$. We have that $f'(s) = (k/2 - 1)f(s)(\log(s/N) + 1) < 0$ for all $s < N/e$, hence $f(s)$ is decreasing for all $s < N/e$. We can now bound the sum (ignoring the constant factor), by splitting it into three types of summands:

- For $s = 2$, we have the term $(2/N)^{k-2}$.
- For $s = 3, 4, \dots, \log^2 N - 1$, we have the upper bound

$$\begin{aligned}
\sum_{s=3}^{\log^2 N-1} (s/N)^{s(k/2-1)} &< (3/N)^{3(k/2-1)} \cdot \log^2 N \\
&= (2/N)^{k-2} \cdot \left[(3/2)^{k-2} \cdot (3/N)^{k/2-1} \cdot \log^2 N\right] \\
&< (2/N)^{k-2} \qquad \text{(for } N \text{ large enough).}
\end{aligned}$$

- For $s = \log^2 N, \ldots, N/e$, we have the upper bound

$$\sum_{s=\log^2 N}^{N/e} (s/N)^{s(k/2-1)} < \left(\frac{\log^2 N}{N}\right)^{\frac{\log^2 N}{N}(k/2-1)} \cdot N/e$$
$$< \left(\frac{1}{N^{\log N}}\right)^{k/2-1} \cdot N/e = \mathsf{negl}(N).$$

Putting together the bounds, we get the desired result. □

A.2 PKE from Sparse LPN

The celebrated work of Applebaum, Barak and Wigderson (ABW) [ABW10] showed how to construct a public-key encryption scheme from k- sparse LPN for a constant $k \geq 3$ (their scheme worked with a constant $k = 3$, but it could be generalized to any constant k). A unique feature of this scheme is that the noise probability could be higher that $o(n^{-0.5})$, which is the feature of Alekhnovich's scheme from the dense variant of LPN [Ale03].

The main idea of the ABW scheme is that the public key is simply a matrix $\mathbf{A} \in \mathbb{F}_2^{n\times m}$ where the columns are chosen at random to be distinct exactly k sparse vectors over $\mathbb{F}_2^n$. The number of samples m is chosen to be a polynomial in n so that $m = n^{1+(k/2-1)(1-\delta)}$ for a tunable constant $\delta > 0$ that is going to affect the noise probabiltiy we choose. The encryption of the message 1 can be set to be a random vector over $\mathbb{F}_2^m$, while to encrypt 0, we simply publish a sparse LPN sample $\boldsymbol{b} = \boldsymbol{s}\mathbf{A} + \boldsymbol{e} \mod 2$. The error here is chosen to be sparse, where each coordinate is chosen to be non-zero with probability $n^{-\delta'}$ for some δ' barely bigger than δ. To decrypt, the main observation is that with $m = n^{1+(k/2-1)(1-\delta)}$, there exist lots of short vectors $\boldsymbol{v}$ of Hamming weight bounded by $\widetilde{O}(n^{\delta})$ so that $\boldsymbol{A}\boldsymbol{v} = 0$. Therefore, such a $\boldsymbol{v}$ can be used to decrypt the ciphertext as $\boldsymbol{b} = \boldsymbol{s}\boldsymbol{A} + \boldsymbol{e}$ when multiplied with $\boldsymbol{v}$ is $\langle \boldsymbol{e}, \boldsymbol{v}\rangle$. When $\boldsymbol{e}$ is chosen sparsely with noise probability $n^{-\delta'}$ for $\delta' > \delta$, for these parameters, this inner product with high probability will be 0. For an encryption of 1, this will be random. ABW then argue that such "secret keys" can be sampled along with $\boldsymbol{A}$.

The beauty of this is that one can choose $\delta > 0$ arbitrarily and work with an arbitrary error rate $n^{-\delta}$, that could be much higher than Alekhnovich's error probability. In a (not so) surprising coincidence, [FKO06] showed that under the same parameter for which the ABW scheme works, we can construct non-deterministic polynomial time refutations for sparse LPN. In this setting, we want to give a short polynomial sized certificate for a ciphertext $\boldsymbol{b}$ being random, and being far from the LPN distribution $\boldsymbol{s}\boldsymbol{A} + \boldsymbol{e}$. For our HSS constructions, we can work with an error probability $n^{-\delta}$ for arbitrary $\delta > 0$ and at the same time our sample complexity could be set to be much smaller than $n^{1+(k/2-1)(1-\delta)}$ assuming k is a large enough constant (or an arbitrary polynomial in n if $k = \omega(1)$). These parameters are neither known to imply PKE nor non-deterministic polynomial time refutations.

B HSS construction from FHE

We give a simple construction of an HSS scheme with $(1 - t/N)$ download rate, assuming an FHE scheme with certain properties. In particular, let FHE be a FHE scheme satisfying:

- ct and s are vectors in some finite field $\mathbb{F}_q$,
- the decryption $\mathsf{Dec}(\mathsf{ct}, s)$ is linear in s and outputs $y\Delta + e$, where e is noise of at most polynomial norm, and Δ is a scaling factor larger than the noise,

- the output y may contain up to $(\log q - \log \Delta)$ bits.

Using such an FHE scheme, we get the following simple HSS construction:

1. An input $x \in \mathbb{F}_q$ is split into shares $\{(\mathsf{ct}(x), s_j)\}_{j\in[N]}$ where $\mathsf{ct}(x)$ is an FHE ciphertext of x and $(s_1, \cdots, s_N)$ is a secret sharing of the secret key s using an arbitrary linear secret sharing scheme LSS over $\mathbb{F}_q$.
2. To compute a function f, each server homomorphically evaluate f on $\mathsf{ct}(x)$ to obtain a ciphertext of the output $\mathsf{ct}(y)$, followed by $y_j = \mathsf{Dec}(\mathsf{ct}(y), s_j)$.
3. Upon collecting all output shares $\{y_j\}_j$, one obtains the noisy output $y\Delta + e$ using the reconstruction procedure Rec of LSS as follows:

$$\begin{aligned}\mathsf{Rec}(y_1, \cdots, y_N) &= \mathsf{Rec}(\mathsf{Dec}(\mathsf{ct}(y), s_1), \cdots, \mathsf{Dec}(\mathsf{ct}(y), s_N)) \\ &= \mathsf{Dec}(\mathsf{ct}(y), \mathsf{Rec}(s_1, \cdots s_N)) = \mathsf{Dec}(\mathsf{ct}(y), s) = y\Delta + e\end{aligned}$$

Since this scheme is compatible with any LSS and LMSS, when instantiated with the multi-secret Shamir sharing [FY92], it achieves a rate of $1 - t/N$. The only caveat, like our HSS scheme, is that the output of this scheme is also noisy. Nevertheless, the ratio between the bit length of y and $\log q$ approaches 1 if $\log q$ is sufficiently larger than $\log e = O(\log \lambda)$.