

Multiparty Homomorphic Secret Sharing From Distributed Discrete Logarithm

Pierre Meyer* Claudio Orlandi† Lawrence Roy‡ Peter Scholl§

Abstract

Homomorphic secret sharing (HSS) enables non-interactive distributed evaluation of functions on secret-shared inputs. While two-party HSS is well understood, multiparty constructions remain limited in expressiveness and efficiency.

We present a general framework for constructing multiparty HSS from two-party semi-private HSS (which in turn can be instantiated using distributed discrete logarithms) via a new *party virtualization* paradigm. In a semi-private HSS, some designated inputs may be fully known to specific servers, while the remaining inputs remain hidden. Starting from two-party semi-private schemes, we obtain N -party semi-private HSS supporting compositions of polynomial computations and restricted multiplication straight-line programs, for $N = O(\log \lambda / \log \log \lambda)$.

We further upgrade semi-private HSS to fully private HSS for functions of polynomial degree and polynomially bounded Waring rank. As a consequence, we obtain multiparty distributed point functions with key size polynomial in λ and $\log |D|$, for a domain D of superpolynomial size. When the underlying two-party scheme supports offline/online sharing, the resulting multiparty DPFs are programmable.

Our framework admits instantiations under standard assumptions including DCR and class-group assumptions.

Contents

1	Introduction	3
1.1	Related Work	4
2	Technical Overview	5
3	Preliminaries	8
3.1	Homomorphic Secret Sharing	8
3.1.1	(Private) HSS.	10
3.1.2	Semi-private HSS.	11
3.2	Waring Rank	12
4	From two-party to multiparty semi-private HSS	13
4.1	The reduction from two-party to multiparty	13
4.2	Instantiations from DCR and Class Groups	18

*CISPA Helmholtz Centre for Information Security, Germany. pierre.meyer@cispa.de

†Aarhus University, Denmark. orlandi@cs.au.dk

‡IBM Research Zürich, Switzerland. ldr709@gmail.com

§Aarhus University, Denmark. peter.scholl@cs.au.dk

5	From semi-private HSS to HSS	20
5.1	From semi-private HSS for $\mathcal{C}_d$ to private HSS for functions of polynomially bounded degree and (mixed) Waring rank	21
5.2	An alternative, committee-based, construction	24
5.3	Applications to multiparty Distributed Point Functions	26
A	Defining (semi-private) “high-end” HSS	30
A.1	(Private) “high-end” HSS.	30
A.2	Semi-private “high-end” HSS.	32
B	Building multiparty “high-end” HSS	34
B.1	Semi-private “high-end” HSS: from two-party to multi-party	35
B.2	From semi-private to private “high-end” HSS	38
C	Deferred proofs	43
C.1	Proof of Corollary 11	43
C.2	Proof of Lemma 17	44
C.3	Proof of Theorem 19	44

1 Introduction

Homomorphic secret sharing (HSS), introduced in [BGI16a], is a powerful primitive that enables a dealer to distribute secret shares of an input among multiple servers such that each party can locally evaluate a function on its share, and the resulting outputs can be combined to recover the value of the function on the original input. Informally, HSS allows distributed computation without interaction during evaluation, and can therefore be viewed as a distributed analogue of fully homomorphic encryption (FHE) [Gen09].

HSS has emerged as a central building block in modern cryptography. On top of the natural use-case of secure multiparty computation, it has found applications in constrained pseudorandom functions [CMPR23, BMO⁺25], trapdoor hash functions [DGI⁺19] and more. The dual primitive of function secret sharing [BGI15] has also been instrumental in the advance of *distributed point functions* [GI14, BGI16b], which can be seen as HSS for the class of point functions, with applications like two-server private database systems [CBM15, WYG⁺17] and private set intersection [DRRT18].

Over the past decade, substantial progress has been made in understanding two-party HSS and FSS, with constructions supporting increasingly expressive classes of functions, and with improvements in the communication overhead of HSS [BGI17, RS21, OSY21, ILL25a]. In contrast, multiparty HSS remains much more elusive, with few notable exceptions (as discussed below). In particular, existing results in multiparty HSS have significant limitations in terms of assumptions, expressiveness, concrete efficiency and the number of supported parties.

Our contribution.

In this work, we present a new approach to building multiparty HSS based on a party virtualization paradigm. Our framework lifts two-party constructions into the multiparty setting, yielding both standard HSS as well as more sophisticated variants supporting additional features like offline/online input sharing. Compared to previous multiparty constructions from group-based assumptions, our framework allows for more expressive function classes and in some cases, a larger number of parties. As an application, we obtain the first group-based constructions of multiparty DPFs with a superpolynomial domain and keys of size $\text{poly}(\lambda, \log |D|)$, for any constant number of parties (or $N = \lambda^{o(1)}$ under subexponential hardness). For any polynomial number of parties, we obtain DPFs with polynomial domain M and keys of size $\text{poly}(\lambda) \cdot M^\varepsilon$ for any $\varepsilon > 0$.

Our starting point is the notion of *semi-private HSS*, recently introduced by [MORS25] (and independently, implicit in [ILL25b]), which can be constructed from distributed discrete logarithm (DDLog) techniques such as used with the DCR assumption or class groups [OSY21, RS21, ADOS22]. HSS (in its weaker, private-sharing form) consists of a sharing phase, where a dealer distributes shares of private inputs to a set of servers, followed by an evaluation phase, where each server locally evaluates on its share to obtain a shared result. The standard security notion requires that all the inputs remain hidden from any given (unqualified) subset of servers holding the shares. In contrast, in a semi-private HSS, some of the inputs are fully known to some of the servers performing the evaluation.

This relaxation of security allows for constructions supporting more expressive function classes. In particular, [MORS25, ILL25b] obtain a two-party semi-private HSS for functions of the form

$$f(x_1, x_2) = P(x_1) \cdot \text{RMS}(x_2),$$

where P captures arbitrary polynomial computations and RMS denotes *restricted multiplication straightline* programs in which inputs can be multiplied by internal wires, and pairs

of internal wires can only be added together (this captures e.g. NC1). The P computation is applied to the *semi-private* input x_1 (which must be known to one of the servers), while the RMS part is applied to the private input x_2 (hidden from both servers)¹.

Our first result is the following transformation, obtained by our new party virtualization technique. We present the result for a constant number of parties, but in some cases we can support more parties (see Section 2 and Table 1 for details).

Theorem 1 (informal). *Let λ be the security parameter and $N = O(1)$. Assuming the existence of two-party semi-private HSS for $P \cdot \text{RMS}$ functions over a ring $\mathcal{R}$, there exists N -party, semi-private HSS for the class of functions $P \cdot \text{RMS} \cdot \dots \cdot \text{RMS}$ over $\mathcal{R}$.*

Next, we show how to upgrade this to fully private HSS supporting the class of functions of polynomial degree and with polynomially bounded Waring rank over $\mathcal{R}$. Recall that a function $f: \mathcal{R}^n \rightarrow \mathcal{R}$ has Waring rank r if it can be expressed as a sum of r d -th powers of linear functions, where d is the degree of f .

Theorem 2 (informal). *Let $N = O(1)$ and $B = \text{poly}(\lambda)$. Assuming the existence of two-party semi-private HSS for $P \cdot \text{RMS}$ functions over a ring $\mathcal{R}$, there exists N -party HSS for the class of functions over $\mathcal{R}$ with degree and Waring rank upper bounded by B .*

From HSS to DPF. Finally, we show that low-Waring-rank HSS suffices to implement distributed point functions (DPFs) for any number of parties. This transformation follows in the same spirit as [BMO⁺25], who represented point functions as low-Waring-rank polynomials to build a privately puncturable PRF. Through a variant of our main construction, we also show that the resulting DPF is *programmable* [BGIK22], meaning that $N - 1$ of the keys can be sampled independently, before choosing the last key as a function of these.

Corollary 3 (informal). *Let $N = O(1)$. Assuming the existence of two-party semi-private HSS for $P \cdot \text{RMS}$ functions over a ring $\mathcal{R}$, there exist N -party distributed point functions supporting a superpolynomially sized domain D and with key size $\text{poly}(\lambda, \log |D|)$. Furthermore, if the two-party HSS supports offline-online input sharing, the N -party DPF is programmable.*

Instantiations. Plugging in known constructions of semi-private HSS for (bounded-size) integer computations [MORS25, ILL25b], we can instantiate the above transformations under a range of assumptions, including decisional composite residuosity (DCR), or assumptions on class groups.

1.1 Related Work

While two-party HSS and DPFs have seen extensive development, the multiparty setting remains comparatively underexplored. Here, we focus on the dishonest majority setting, where up to $N - 1$ out of N parties may be corrupted.

For HSS, the first multi-party constructions were based on indistinguishability obfuscation [BGI15] and later from learning with errors via spooky encryption [DHRW16], an advanced form of multi-key fully homomorphic encryption. The work of Chillotti et al. [COS⁺22, COSL24] presented several simplifications and improvements over the spooky approach, obtaining simpler constructions from LWE, as well as the first *four-party* HSS using group-based assumptions. In particular, they observed that in prior two-party constructions based

¹What we described is the notion used in [MORS25, ILL25b]. In this paper however, all of our constructions would also work with a weakened version where x_2 *can* but does not *have to* be revealed to the second server. This is in contrast with the first input, which *must* be known to the first server so it can run the correct homomorphic evaluation procedure.

		No Committees	Committees
Polynomial Hardness	Parties:	$O(1)$	$N = \text{poly}(\lambda)$
	Degree:	$\text{poly}(\lambda)$	$O(1)$
	DPF Domain:	$M \leq 2^{\text{poly}(\lambda)}$	$M \leq \text{poly}(\lambda)$
	DPF Key Size:	$\text{poly}(\lambda, \log M)$	$\text{poly}(\lambda) \cdot M^\varepsilon \quad \forall \varepsilon > 0$
Subexponential Hardness	Parties:	$O(\log \lambda / \log \log \lambda)$	$N = \text{poly}(\lambda)$
	Degree:	$\text{poly}(\lambda)$	$O(\log \lambda / (\log N + \log \log \lambda))$
	DPF Domain:	$M \leq 2^{\text{poly}(\lambda)}$	$M \leq 2^{O((\log \lambda)^2 / (\log N + \log \log \lambda))}$
	DPF Key Size:	$\text{poly}(\lambda, \log M)$	$\text{poly}(\lambda) \cdot M^{(\log N + \log \log \lambda) / \log \lambda}$

Table 1: Parameter regimes achieved by our construction.

on DCR, the evaluation function for a single homomorphic multiplication is an NC1 function. This allows the two-party HSS evaluation to be evaluated recursively, obtaining a four-party HSS for constant-depth circuits. Later, Boyle, Couteau and Meyer [BCM23] improved this result using complexity leveraging, obtaining four-party HSS for functions computable by circuits of depth $\log \log \lambda$.

Abram, Roy and Scholl [ARS24] introduced the notion of *succinct HSS* in the two-party setting, with applications to sublinear communication MPC and multiparty distributed point functions. While their DPF constructions allow to generate a large batch of DPF keys with communication cost sublinear in the batch size, they are limited to point functions with a domain of size $D = \text{poly}(\lambda)$, and the resulting key size still grows with $\Omega(\sqrt{D})$.

Dao et al. [DIJL23] presented a multiparty HSS based on a sparse variant of the learning parity with noise assumption, supporting homomorphic evaluation of low-degree polynomials for any polynomial number of parties. However, their construction is inherently noisy, and in particular does not imply a multiparty distributed point function.

Very recently, Couteau et al. [CKY25] showed that assuming subexponential hardness of LPN and MQ, one can achieve $\log^* \lambda$ -party HSS for circuits of depth $\log^* \lambda$. When applied to building DPFs, they obtain a multi-party DPF for any constant number of parties with superpolynomial domain size.

Compared to these works, our approach differs conceptually and technically. We are the first to achieve HSS and DPFs for any constant number of parties solely from group-based assumptions, or through party virtualization techniques. When allowing subexponential hardness assumptions, we even obtain DPFs that support a superpolynomially sized domain and $N = \lambda^{o(1)}$ parties, which was only previously known using lattice-based assumptions.

2 Technical Overview

N-party semi-private HSS via party virtualization.

Our main technical contribution is a general compilation technique that lifts a two-party sp-HSS to a N -party sp-HSS via a process akin to *party virtualization* (see Fig. 1). At a high level, we recursively combine a $(N - 1)$ -party sp-HSS with a two-party sp-HSS by using the latter to evaluate the HSS algorithms of the former. That is, the two-party sp-HSS is used to securely emulate one virtual party inside the $(N - 1)$ -party protocol. This yields a N -party sp-HSS for functions of the form

$$f(x_1, \dots, x_N) = P(x_1) \cdot \text{RMS}(x_2) \cdots \text{RMS}(x_N).$$

As a warm-up, suppose we have a two-party semi-private HSS for $P \cdot \text{RMS}$, consisting of a sharing algorithm $\text{Share}_2(x_1, x_2)$, where x_1 is the semi-private input and x_2 the private

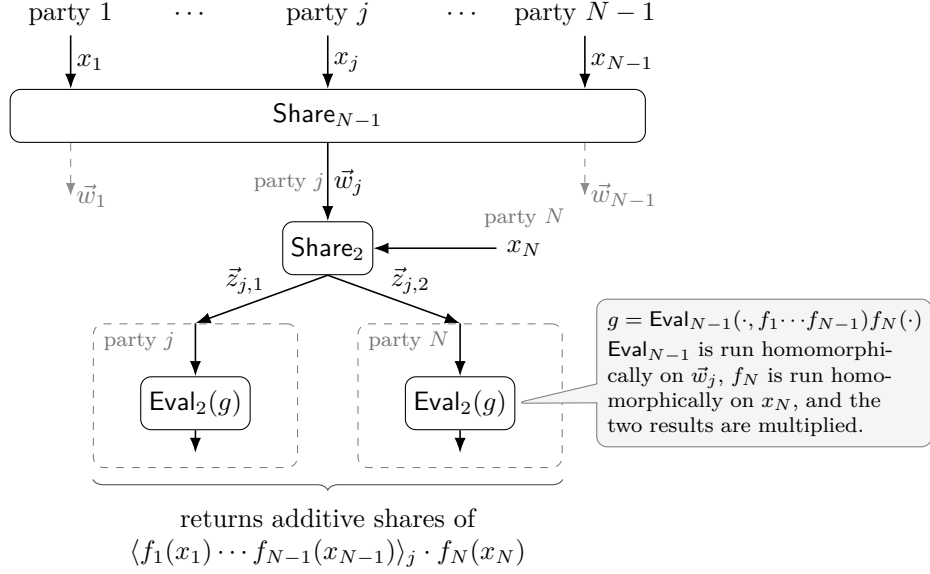


Figure 1: The first $N - 1$ parties share their inputs using the $(N - 1)$ -party scheme. Then each one re-shares their resulting share, but this time pairwise with the N^{th} party. They then pairwise (each $j \in [1, N - 1]$ with party N) homomorphically evaluates Eval_{N-1} inside Eval_2 . Only party j 's branch is drawn, but summing all output shares over j gives $f_1(x_1) \cdots f_N(x_N)$.

input, together with an evaluation algorithm Eval_2 . We can construct a 3-party semi-private HSS for $\text{P} \cdot \text{RMS} \cdot \text{RMS}$ as follows.

Sharing: To share inputs x_1, x_2, x_3 , we first run $\text{Share}_2(x_1, x_2)$ to obtain shares $\vec{w}_1$ and $\vec{w}_2$ of $\vec{w} = x_1 \| x_2$. Then, we reshare these shares by running $\text{Share}_2(\vec{w}_i, x_3)$ to get shares $(\vec{z}_{i,1}, \vec{z}_{i,2})$, for $i \in \{1, 2\}$. Finally, we output

$$\begin{aligned} \text{share}_1 &= \vec{z}_{1,1} \\ \text{share}_2 &= \vec{z}_{2,1} \\ \text{share}_3 &= (\vec{z}_{1,2}, \vec{z}_{2,2}) \end{aligned}$$

Note that, by the semi-private property, share $z_{i,1}$ may leak information on $\vec{w}_i$, and similarly, $\vec{w}_1$ may leak on input x_1 . This is not an issue, though, since x_1 is a semi-private input, so leaking it to party 1 via the share $\vec{w}_1$ does not compromise semi-private security. On the other hand, leaking $\vec{w}_2$ to party 2 reveals nothing, as $\vec{w}_2$ is itself an input-hiding share.

Evaluation: To evaluate on a function $f_1(x_1) \cdot f_2(x_2) \cdot f_3(x_3)$, party i proceeds as follows: Using Eval_2 , parties $i \in \{1, 2\}$ perform a homomorphic evaluation of the function

$$(x, y) \mapsto \text{Eval}_2(x, f_1 \cdot f_2) \cdot f_3(y)$$

on their share $\vec{z}_{i,1}$.

Meanwhile, party 3 performs a homomorphic evaluation of the same function, once on each of its shares $\vec{z}_{1,2}$ and $\vec{z}_{2,2}$, and adds the results together.

The “inner” Eval_2 functions perform the homomorphic evaluation of $f_1 \cdot f_2$ on the shares $\vec{z}_{i,1}$, leading to a share which is then multiplied by f_3 in the “outer” evaluation. This ends up with a sharing the desired result, $f_1(x_1) \cdot f_2(x_2) \cdot f_3(x_3)$.

Notice that each of the Eval_2 functions is only used on functions in the class $\text{P} \cdot \text{RMS}$, since the Eval_2 algorithm itself is in P , and is always applied to the semi-private input.

Going beyond two parties.

Above, we obtained a 3-party semi-private HSS for $P \cdot \text{RMS} \cdot \text{RMS}$ from a two-party semi-private HSS for $P \cdot \text{RMS}$. Generalising and iterating this process, we can obtain a N -party semi-private HSS for $P \cdot \text{RMS} \cdots \text{RMS}$, where the first input is semi-private (known to the first party), and the remaining inputs are hidden from all parties. Since each iteration blows up the computation by a polynomial factor, for now we are limited to multi-party HSS for $N = O(1)$ parties.

From semi-private to fully private HSS.

We show how to upgrade the security of our semi-private HSS to fully private HSS, albeit for a more restricted class of functions. Concretely, we want to implement N -party HSS for the class of functions with bounded Waring rank, i.e. that can be written as:

$$f(x) = \sum_{i=1}^r A_i(x)^{d_i},$$

where A_i are affine function over $\mathbb{Z}$, $d_i = \text{poly}(\lambda)$ and $r = \text{poly}(\lambda)$ is the Waring rank of f .

To bypass the semi-private limitation of our initial HSS, a natural starting point is to additively share the input x into $x_1 + \cdots + x_N$, and feed each share x_i as an input to the semi-private HSS. Now, even if semi-private security means x_i is leaked to the i -th server, the underlying x remains hidden.

The main challenge, however, is to express the function $f(x)$ as a product of functions of the shares, i.e. into the form $P(x_1) \cdot \text{RMS}(x_2) \cdots \text{RMS}(x_N)$. When starting with HSS over $\mathbb{Z}$, we could not see a way to make this translation. However, if we instead start with an HSS supporting computations over a *matrix ring* $\mathbb{Z}^{m \times m}$, we are able to make it work.

We explain the approach for a simplified case where $f(x) = (x_1 + \cdots + x_N)^d$ is a d -th power of a single linear function. Extending to the more general case with multiple affine functions is then straightforward. First, encode each share x_i as a matrix

$$X_i = \left(\begin{array}{c|c} x_i & 0 \\ \hline 0 & 0 \end{array} \right) \in \mathbb{Z}^{(d+1) \times (d+1)}.$$

The idea is to iteratively compute $f(x)$ by defining a sequence of polynomials starting with $f_0(t) = t^d$ and $f_i(t) = f_{i-1}(t + x_i)$ for $i \in [N]$, which gives $f_N(0) = (x_1 + \cdots + x_N)^d$. Viewing $f_i(t)$ as a coefficient vector of the corresponding polynomial in t , by the binomial theorem it can be seen that the substitution $t \mapsto t + x_i$ applies a linear map to the coefficients. This map is given by the following matrix $M(y)$:

$$M(y) := \begin{pmatrix} 1 & y & y^2 & \cdots & y^d \\ 0 & 1 & \binom{2}{1}y & \cdots & \binom{d}{1}y^{d-1} \\ \vdots & 0 & 1 & \cdots & \binom{d}{2}y^{d-2} \\ & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

It can be shown from the binomial theorem that these matrices satisfy a multiplicative property, $M(y_1)M(y_2) = M(y_1 + y_2)$. It follows that, if we use N -party semi-private HSS to compute $Y = \prod_{i=1}^N M(x_i) = M(x)$, we can extract $f(x)$ from the top-right entry of Y .

From (private) HSS to DPF.

Our bounded Waring rank HSS suffices to implement N -party distributed point functions (DPFs) with keys of size polylogarithmic in the domain size. This follows immediately from the observation from [BMO⁺25] that point functions can be represented as bounded Waring rank polynomials.

More advanced forms of HSS and DPF.

The above construction yields what we refer to as *low-end HSS*, namely the notion where all inputs are shared at once by a dealer. In Appendix B, we also present a transformation for a stronger form of *high-end HSS*, where inputs can be shared individually, and shares additionally support an *offline/online* decomposition (where some shares can be sampled before the corresponding input is known).

In this case, our virtualization framework preserves the offline/online property: starting with a 2-party semi-private, “high-end” HSS, we obtain a N -party semi-private HSS that also supports offline/online sharing, namely, where up to $N - 1$ shares of every input can be sampled ahead of time. This offline/online property also extends to the final *private* N -party HSS for functions of bounded Waring rank, which further implies *programmable* N -party DPFs.

Beyond a constant number of parties.

We offer a couple of ways to overcome the limitation of our nested, two-to- N party construction, obtaining HSS and/or DPFs for a super-constant number of parties. Firstly, for functions where the degree d is constant, we can adopt a *committee-based* approach. The idea is that, since in the expansion of $(y_1 + \dots + y_N)^d$, each monomial depends on at most d variables, we can run an independent d -party semi-private HSS for each subset $S \subseteq [N]$ of size at most d . This supports polynomially many parties $N = \text{poly}(\lambda)$ as long as the degree is $d = O(1)$ (otherwise the number of committees blows up).

Additionally, by employing complexity leveraging techniques and relying on subexponential hardness of the underlying HSS scheme, the recursive approach can be run with up to $O(\log \lambda / \log \log \lambda)$ parties while supporting any $d = \text{poly}(\lambda)$. Combining this with the committee approach, we can alternatively support any $N = \text{poly}(\lambda)$ and a super-constant function degree.

The resulting tradeoffs and their implications for multi-party DPFs are summarized in Table 1.

3 Preliminaries

Notation for additive shares. If $S \subseteq [n]$ is a non-empty set of parties, we use the notation $\langle x \rangle_S$ to signify that the value x is split into $|S|$ additive shares. For each $i \in S$, we further use the notation $\langle x \rangle_S^{(i)}$ to denote party i ’s share of x . Note that the relation $x = \sum_{i \in S} \langle x \rangle_S^{(i)}$ will, depending on the context, either hold over $\mathbb{Z}$ or only over an additive subgroup of $\mathbb{Z}$.

3.1 Homomorphic Secret Sharing

A Homomorphic Secret Sharing (HSS) scheme allows for one or more secret values to be shared (usually not additively) while supporting the local evaluation of functions on the shares (usually resulting in additive “post-homomorphism” output shares). Recent

works [MORS25] have shown that the homomorphic expressivity can be extended significantly if we relax input privacy, and require that each of the inputs is known to exactly one of the parties: we refer to such inputs as *semi-private* (note that the joint evaluation of multiple semi-private inputs is typically known to no party).

The main construction of this paper is a somewhat generic reduction from two-party semi-private HSS for some class to d -party semi-private HSS for some related class. However, there is no unified definition capturing all known variants of HSS (with or without setup, secret-key or public-key, staged/offline-online or not...) which can be lifted from the two-party setting to the multiparty setting using this technique. We therefore focus our efforts on exemplify to usefulness of our reduction in two extremes cases:

1. **“Low-end” HSS.** We put forward a first definition (Definition 5), which is as weak as possible² while still allowing the resulting d -party HSS to imply d -DPFs. We posit such minimality should help simplify the exposition, and make it easier for future constructions to instantiate the required 2-HSS (even if it means the resulting d -HSS is weakened in the same way).
2. **“High-end” HSS.** We put forward a second definition (Definition 22), which additionally captures some of the properties satisfied by Meyer, Orlandi, Roy, and Scholl’s [MORS25] construction of 2-HSS. These extra properties yield a much stronger form of d -HSS and, in turn, of d -DPF.

The body of this paper is dedicated to the “low-end” HSS (hereafter, simply “HSS” by default), and we refer to the appendix for explanations on how our transforms can be adapted to “high-end” HSS. Specifically, “low-end” HSS is defined in Section 3.1.1, “low-end” semi-private HSS in Section 3.1.2, “high-end” (private) HSS in Appendix A.1, and “high-end” semi-private HSS in Appendix A.2.

Admissible inputs. An annoying technicality plaguing many known constructions of HSS (including the ones presented in this paper) is that correctness of reconstruction is only guaranteed provided we compute on small enough values. Most notably, many group-based schemes use a “ B -bounded” model of computation, where we must ensure that every intermediary value computed (*e.g.* by an addition or a multiplication) in the course of the homomorphic evaluation must remain below a certain bound B . The generally agreed upon solution is to embed our computation in a much larger ring, and restrict ourselves to small enough inputs that the value of every computation gate is bounded. This solution affects little beyond the scheme’s rate and concrete efficiency, and we therefore adopt it in this paper.

A class of computations $\mathcal{C}$ may come with, for each $C \in \mathcal{C}$, a set of C -*admissible* inputs, *i.e.* a subset of the domain of C on which correctness is required to hold. Typical examples are the B -bounded inputs of integer computations (Theorem 12), and the restriction of every input to a subset $S \subseteq \mathcal{R}$ of the ring (Definition 8). Unless specified otherwise, every input is admissible. Admissibility only restricts correctness: security (in all four definitions below) is required for all inputs in the domain, since the corrupted parties’ inputs are handed to the simulator anyway and an honest party’s input should remain hidden even if it is not admissible.

²This informal statement should not be taken for a claim of minimality in the foundational sense.

3.1.1 (Private) HSS.

In this section we define what is arguably one of the most bare-bones forms of homomorphic secret sharing.

Definition 4 (HSS). Let $\ell \in \mathbb{N} \setminus \{0, 1\}$ be a number of parties. Let $\mathcal{R} = \mathcal{R}(\lambda)$ be a ring, and let $n \in \text{poly}(\lambda)$ be an input size. An ℓ -party additive HSS scheme supporting a class of computations $\mathcal{C} \subseteq \mathcal{R}^n \rightarrow \mathcal{R}$ is a pair of polynomial-time algorithms $\text{HSS} = (\text{Share}, \text{Eval})$, where **Share** is probabilistic but **Eval** is deterministic³, with the following syntax:

- $\text{HSS.Share}(1^\lambda, (y_1, \dots, y_n))$: On input the security parameter 1^λ and an n -tuple of inputs $(y_1, \dots, y_n) \in \mathcal{R}^n$, the input sharing algorithm **Share** outputs an ℓ -tuple of input shares $((y_1^{(1)}, \dots, y_n^{(1)}), \dots, (y_1^{(\ell)}, \dots, y_n^{(\ell)}))$.
- $\text{HSS.Eval}(j, (x_1^{(j)}, \dots, x_n^{(j)}), C)$: On input a party index j , an input share $(x_i^{(j)})_{i \in [n]}$, and a supported evaluation circuit $C \in \mathcal{C}$, the homomorphic evaluation algorithm **Eval** returns the j^{th} output share $y^{(j)} \in \mathcal{R}$.

We additionally require that HSS satisfy the following notion of correctness, and one of the two notions of security. If it only satisfies the relaxed notion of *semi-private* security, we say HSS is a *semi-private* HSS scheme.

- **Correctness.** For any sufficiently large $\lambda \in \mathbb{N}$, for any circuit $C \in \mathcal{C}$, any C -admissible inputs $(x_1, \dots, x_n) \in \mathcal{R}^n$,

$$\Pr \left[y = C(x_1, \dots, x_n) : y \xleftarrow{\$} \text{Exp}_{\text{corr}}^{\text{HSS}}(1^\lambda, x_1, \dots, x_n) \right] \geq 1 - \text{negl}(\lambda)$$

where $\text{Exp}_{\text{corr}}^{\text{HSS}}$ is defined in Fig. 2.

- **Security.** There exists a *p.p.t.* algorithm **Sim** (a simulator) such that, for every set of corrupted parties $\mathcal{Z} \subsetneq [\ell]$ and for any inputs $(x_1, \dots, x_n) \in \mathcal{R}^n$ (admissible or not), the output of the experiments **Real_Z** and **Ideal_Z** (Fig. 3) are computationally indistinguishable.

Experiment $\text{Exp}_{\text{corr}}^{\text{HSS}}$

$\text{Exp}_{\text{corr}}^{\text{HSS}}(1^\lambda, x_1, \dots, x_n) : //$ Correctness Experiment

1. $((x_1^{(1)}, \dots, x_n^{(1)}), \dots, (x_1^{(\ell)}, \dots, x_n^{(\ell)})) \xleftarrow{\$} \text{Share}(1^\lambda, (x_1, \dots, x_n))$
2. For $j \in [\ell]$, $z^{(j)} \leftarrow \text{Eval}(j, (x_1^{(j)}, \dots, x_n^{(j)}), C)$
3. Output $z \leftarrow \sum_{j \in [\ell]} z^{(j)}$

Figure 2: Correctness of the notion of HSS of Definition 4

³Following the convention that **Eval** should be deterministic is more convenient in settings such as this paper's, where **Eval** is itself evaluated homomorphically in a recursive fashion.

Experiment $\text{Real}_{\mathcal{Z}}$ and $\text{Ideal}_{\mathcal{Z}}$

$\text{Real}_{\mathcal{Z}}(1^\lambda, x_1, \dots, x_n) : // \text{ Security: Real World}$

1. $((x_1^{(1)}, \dots, x_n^{(1)}), \dots, (x_1^{(\ell)}, \dots, x_n^{(\ell)})) \xleftarrow{\$} \text{Share}(1^\lambda, (x_1, \dots, x_n))$
2. Output $(x_1^{(j)}, \dots, x_n^{(j)})_{j \in \mathcal{Z}}$

$\text{Ideal}_{\mathcal{Z}}(1^\lambda, n) : // \text{ Security: Ideal World}$

Output $\text{Sim}(1^\lambda, \mathcal{Z}, n)$

Figure 3: Experiments for security in the notion of HSS of Definition 4

3.1.2 Semi-private HSS.

We use HSS with a relaxed notion of privacy: each of the shared values need only be private from all-but-one parties.

Definition 5 (Semi-private HSS). Let $\ell \in \mathbb{N} \setminus \{0, 1\}$ be a number of parties. Let $\mathcal{R} = \mathcal{R}(\lambda)$ be a ring and $n_1, \dots, n_\ell \in \text{poly}(\lambda)$. An ℓ -party semi-private additive HSS scheme supporting a class of computations $\mathcal{C} \subseteq \mathcal{R}^{n_1} \times \dots \times \mathcal{R}^{n_\ell} \rightarrow \mathcal{R}$ is a pair of polynomial-time algorithms $\text{HSS} = (\text{Share}, \text{Eval})$, where Share is probabilistic but Eval is deterministic, with the following syntax:

- $\text{HSS.Share}(1^\lambda, (y_1, \dots, y_\ell))$: On input the security parameter 1^λ and an ℓ -tuple of semi-private inputs $(y_1, \dots, y_\ell) \in \mathcal{R}^{n_1} \times \dots \times \mathcal{R}^{n_\ell}$, the semi-private input sharing algorithm Share outputs an ℓ -tuple of input shares $((y_1^{(1)}, \dots, y_\ell^{(1)}), \dots, (y_1^{(\ell)}, \dots, y_\ell^{(\ell)}))$.
- $\text{HSS.Eval}(j, (x_1^{(j)}, \dots, x_\ell^{(j)}), C)$: On input a party index j , an input share $(x_i^{(j)})_{i \in [\ell]}$, and a supported evaluation circuit $C \in \mathcal{C}$, the homomorphic evaluation algorithm Eval returns the j^{th} output share $y^{(j)} \in \mathcal{R}$.

We additionally require correctness and *semi-private* security, as follows.

- **Correctness.** For any sufficiently large $\lambda \in \mathbb{N}$, for any circuit $C \in \mathcal{C}$, any C -admissible inputs $(x_1, \dots, x_\ell) \in \mathcal{R}^{n_1} \times \dots \times \mathcal{R}^{n_\ell}$,

$$\Pr \left[y = C(x_1, \dots, x_\ell) : y \xleftarrow{\$} \text{Exp}_{\text{corr}}^{\text{HSS}}(1^\lambda, x_1, \dots, x_\ell) \right] \geq 1 - \text{negl}(\lambda)$$

where $\text{Exp}_{\text{corr}}^{\text{HSS}}$ is defined in Fig. 4.

- **Semi-Privacy.** There exists a *p.p.t.* algorithm Sim^{sp} (a simulator) such that, for every set of corrupted parties $\mathcal{Z} \subseteq [\ell]$ and for any inputs $(x_1, \dots, x_\ell) \in \mathcal{R}^{n_1} \times \dots \times \mathcal{R}^{n_\ell}$ (admissible or not), the output of the experiments $\text{Real}_{\mathcal{Z}}^{\text{sp}}$ and $\text{Ideal}_{\mathcal{Z}}^{\text{sp}}$ (Fig. 5) are computationally indistinguishable.

Experiment $\text{Exp}_{\text{corr}}^{\text{HSS}}$

$\text{Exp}_{\text{corr}}^{\text{HSS}}(1^\lambda, x_1, \dots, x_\ell) : //$ Correctness Experiment

1. $((x_1^{(1)}, \dots, x_\ell^{(1)}), \dots, (x_1^{(\ell)}, \dots, x_\ell^{(\ell)})) \xleftarrow{\$} \text{Share}(1^\lambda, (x_1, \dots, x_\ell))$
2. For $j \in [\ell]$, $z^{(j)} \leftarrow \text{Eval}(j, (x_1^{(j)}, \dots, x_\ell^{(j)}), C)$
3. Output $z \leftarrow \sum_{j \in [\ell]} z^{(j)}$

Figure 4: Correctness of the notion of HSS of Definition 5

Experiment $\text{Real}_{\mathcal{Z}}^{\text{sp}}$ and $\text{Ideal}_{\mathcal{Z}}^{\text{sp}}$

$\text{Real}_{\mathcal{Z}}^{\text{sp}}(1^\lambda, x_1, \dots, x_\ell) : //$ Semi-privacy: Real World

1. $((x_1^{(1)}, \dots, x_\ell^{(1)}), \dots, (x_1^{(\ell)}, \dots, x_\ell^{(\ell)})) \xleftarrow{\$} \text{Share}(1^\lambda, (x_1, \dots, x_\ell))$
2. Output $(x_1^{(j)}, \dots, x_\ell^{(j)})_{j \in \mathcal{Z}}$

$\text{Ideal}_{\mathcal{Z}}^{\text{sp}}(1^\lambda, (n_i)_{i \in [\ell]}, (x_j)_{j \in \mathcal{Z}}) : //$ Semi-privacy: Ideal World

Output $\text{Sim}^{\text{sp}}(1^\lambda, \mathcal{Z}, (n_i)_{i \in [\ell]}, (x_j)_{j \in \mathcal{Z}})$

Figure 5: Experiments for semi-private security in the notion of HSS of Definition 5

3.2 Waring Rank

A function has polynomially bounded Waring rank if it can be expressed as the sum of at most polynomially many powers of linear functions, and has polynomial degree. As demonstrated by [BMO⁺25], such a decomposition can prove useful when evaluating a function in HSS.

Definition 6 (Waring Rank and its variants). Let $\mathcal{R}$ be a commutative ring and let $f \in \mathcal{R}[X_1, \dots, X_N]$ be a polynomial of degree $d \in \mathbb{N}$. In the following, let $f_1, \dots, f_r \in \mathcal{R}[X_1, \dots, X_N]$ denote affine functions⁴ and $\alpha_1, \dots, \alpha_r \in \mathcal{R}$.

- The *Waring rank* of f is defined to be the smallest integer $r \in \mathbb{N}$ such that there exist affine functions $f_1, \dots, f_r$ and coefficients $\alpha_1, \dots, \alpha_r$ satisfying

$$f(X_1, \dots, X_N) = \sum_{i=1}^r \alpha_i (f_i(X_1, \dots, X_N))^d.$$

We will refer to the set $\{(f_1, \alpha_1), \dots, (f_r, \alpha_r)\}$ as a Waring decomposition of f of rank r .

- The *Mixed-degree Waring rank* of f is defined to be the smallest integer $r \in \mathbb{N}$ such

⁴We allow affine (instead of linear) functions here because we are concerned with representing inhomogeneous polynomials.

that there exist affine functions $f_1, \dots, f_r$ and coefficients $\alpha_1, \dots, \alpha_r$ satisfying

$$f(X_1, \dots, X_N) = \sum_{i=1}^r \alpha_i (f_i(X_1, \dots, X_N))^{d_i}, \quad \text{where each } 0 \leq d_i \leq d.$$

We will refer to the set $\{(f_1, d_1, \alpha_1), \dots, (f_r, d_r, \alpha_r)\}$ as a Mixed Waring decomposition of f of rank r .

Remark 7 (Waring decompositions over rings). The Waring decompositions of [BMO⁺25] are stated over the field $\mathbb{Q}$. However, any such decomposition whose coefficients have denominators that are invertible in $\mathcal{R}$ yields (by mapping each coefficient to $\mathcal{R}$) a decomposition over $\mathcal{R}$ of the same rank and with the same exponents d_i . In this paper we only ever divide by integers at most d , so they are valid over any commutative ring in which $1, \dots, d$ are invertible, *e.g.* any field of characteristic 0 or greater than d .

4 From two-party to multiparty semi-private HSS

In this section, we define the class of “ ℓ -split functions” $\mathcal{C}_\ell$ (Definition 8), then provide (in Fig. 6) a recursive construction of ℓ -party semi-private HSS for $\mathcal{C}_\ell$. Roughly speaking, functions in $\mathcal{C}_\ell$ have ℓ inputs, and output the product of ℓ functions, each applied to a different input. While the first input’s function may be computed by an arbitrary polynomial-size arithmetic circuit, all of the other functions must be computed by a logarithmic-depth arithmetic circuit (or equivalently, by a *Restricted-Multiplication Straight-line program*, or RMS [BGI16a, Definition 3.1]).

In Section 4.1, we provide the generic reduction, and in Section 4.2 we show that the starting point (*i.e.* the two-party semi-private HSS) can be instantiated from DCR or class groups.

4.1 The reduction from two-party to multiparty

Definition 8 (Split functions). Let $\mathcal{R}$ be a ring, let $\ell \in \mathbb{N}$, and let $n_1, \dots, n_\ell \in \mathbb{N}$. We define the class of ℓ -split functions $\mathcal{C}_\ell$ as follows:

$$\mathcal{C}_\ell := \left\{ \begin{array}{ccc} \mathcal{R}^{n_1} \times \dots \times \mathcal{R}^{n_\ell} & \rightarrow & \mathcal{R} \\ (x_1, x_2, \dots, x_\ell) & \mapsto & f_1(x_1)f_2(x_2)\dots f_\ell(x_\ell) \end{array} \mid \begin{array}{l} f_1 \in \mathbf{P} \\ f_2, \dots, f_\ell \in \mathbf{RMS} \end{array} \right\}$$

where $\mathbf{P}$ is the class of all polynomial-size $\mathcal{R}$ -arithmetic circuits (of arbitrary degree), and $\mathbf{RMS}$ denotes the class of restricted-multiplication straight-line programs [BGI16a, Definition 3.1] over $\mathcal{R}$. We sometimes further parameterise $\mathcal{C}_N$ by specifying a finite subset $S \subseteq \mathcal{R}$ of *admissible inputs* (in the sense of Section 3.1). It should then be understood that the admissible inputs for a function in $\mathcal{C}_N$ are restricted to $S^{n_1} \times \dots \times S^{n_N}$.

N-HSS

// N -party semi-private HSS (as per Definition 5) for the class $\mathcal{C}_N$ (Definition 8), as parameterised by input sizes $n_1, \dots, n_N$

Requires:

- The classes $\mathcal{C}_2$, $\mathcal{C}_{N-1}$, and $\mathcal{C}_N$ should all be parameterised by the same subset $S \subseteq \mathcal{R}$ of admissible inputs, or by none at all. If S is defined, we require $\{0_{\mathcal{R}}, 1_{\mathcal{R}}\} \subseteq S$, so that the encoded shares $\tilde{w}_i$ below are admissible inputs of $\text{HSS}_{2,i}$.

- $\text{HSS}_{N-1} = (\text{Share}_{N-1}, \text{Eval}_{N-1})$ is a $(N-1)$ -party semi-private HSS (as per Definition 5) for the class $\mathcal{C}_{N-1}$ (Definition 8), as parameterised by input sizes $n_1, \dots, n_{N-1}$.
- For $i \in [N-1]$, let $M_i(\lambda)$ be a polynomial upper bound on the bit length of the i^{th} share output by $\text{Share}_{N-1}(1^\lambda, \cdot)$ when run on admissible inputs, and let $L_i(\lambda)$ be a polynomial upper bound on the bit length of the output of $\text{Eval}_{N-1}(i, \cdot, \cdot)$ (both exist since these algorithms run in polynomial time). Set $m_i := M_i$, let $\text{Encode}_i: \{0, 1\}^{M_i} \rightarrow \mathcal{R}^{m_i}$, $(b_1, \dots, b_{M_i}) \mapsto (b_1 \cdot 1_{\mathcal{R}}, \dots, b_{M_i} \cdot 1_{\mathcal{R}})$ be the one-bit-per-ring-element encoding, and let $\text{Decode}_i: \{0_{\mathcal{R}}, 1_{\mathcal{R}}\}^{m_i} \rightarrow \{0, 1\}^{M_i}$ be its inverse. Finally, fix constants $\rho_{i,1}, \dots, \rho_{i,L_i} \in \mathcal{R}$ such that every output y of $\text{Eval}_{N-1}(i, \cdot, \cdot)$ has a binary representation $(b_1, \dots, b_{L_i}) \in \{0, 1\}^{L_i}$ with $y = \sum_{k \in [L_i]} b_k \rho_{i,k}$ (e.g. $\rho_{i,k} = 2^{k-1}$ for $\mathcal{R} = \mathbb{Z}$ or $\mathbb{Z}_p$, and $\rho_{i,k} = 2^{k'} E_{u,v}$ for a matrix ring, where $E_{u,v}$ is a matrix unit and k' the position of the bit within the entry).
- For $i \in [N-1]$, $\text{HSS}_{2,i} = (\text{Share}_{2,i}, \text{Eval}_{2,i})$ is a 2-party semi-private HSS (as per Definition 5) for the class $\mathcal{C}_2$ (Definition 8), as parameterised by input sizes m_i, n_N .
- For $i \in [N-1]$, $g_i: \mathcal{R}^{m_i} \rightarrow \mathcal{R}$ is the $\mathcal{R}$ -arithmetic circuit obtained as follows. Let $C_i: \{0, 1\}^{M_i} \rightarrow \{0, 1\}^{L_i}$ be a boolean circuit computing $X \mapsto \text{Eval}_{N-1}(i, \text{Decode}_i(X), (f_1, \dots, f_{N-1}))$, with the output written in binary as above. Emulate each gate of C_i arithmetically ($\neg a \mapsto 1_{\mathcal{R}} - a$, $a \wedge b \mapsto ab$, $a \oplus b \mapsto a + b - 2ab$) and output $\sum_{k \in [L_i]} \rho_{i,k} \cdot C_i(X)_k$. On inputs in $\{0_{\mathcal{R}}, 1_{\mathcal{R}}\}^{m_i}$ every wire of the emulation is $0_{\mathcal{R}}$ or $1_{\mathcal{R}}$, so $g_i(\text{Encode}_i(s)) = \text{Eval}_{N-1}(i, s, (f_1, \dots, f_{N-1}))$ for all $s \in \{0, 1\}^{M_i}$. Note that $g_i \in \mathbf{P}$: it has polynomial size, and no degree restriction applies.

$\text{HSS}_N.\text{Share}(1^\lambda, \vec{x} = (x_1, \dots, x_N)) :$

1. $(w_1, \dots, w_{N-1}) \xleftarrow{\$} \text{Share}_{N-1}(1^\lambda, (x_1, \dots, x_{N-1}))$
2. For $i \in [N-1]$, $\tilde{w}_i \leftarrow \text{Encode}_i(w_i)$
3. For $i \in [N-1]$, $(z_{i,1}, z_{i,2}) \xleftarrow{\$} \text{Share}_{2,i}(1^\lambda, (\tilde{w}_i, x_N))$
4. For $i \in [N-1]$, $\text{share}_i \leftarrow z_{i,1}$
5. $\text{share}_N \leftarrow (z_{j,2})_{j \in [N-1]}$
6. Output $(\text{share}_1, \dots, \text{share}_N)$

$\text{HSS}_N.\text{Eval}(i, \text{share}_i, (f_1, \dots, f_N)) :$

1. If $i \in [N-1]$:

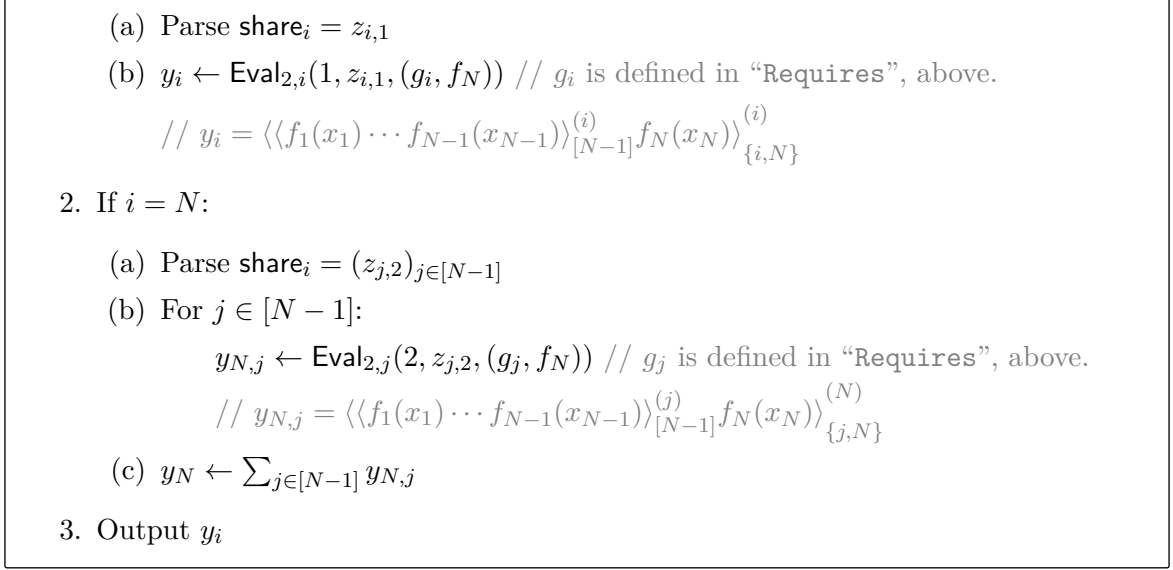


Figure 6: Semi-private N -HSS for $\mathcal{C}_N$ from semi-private $(N-1)$ -HSS for $\mathcal{C}_{N-1}$ and semi-private 2-HSS for $\mathcal{C}_2$.

Theorem 9. *If its requirements are met, then the construction of Fig. 6 is an N -party semi-private HSS (as per Definition 5) for the class $\mathcal{C}_N$ (Definition 8), as parameterised by input sizes $n_1, \dots, n_N$. Moreover, if*

- $S_N(s_1, \dots, s_N)$ (resp. $S_{N-1}(s_1, \dots, s_{N-1})$, resp. $S_2(s_1, s_2)$) denotes the size of the shares output by $\text{HSS}_N.\text{Share}$ (resp. $\text{HSS}_{N-1}.\text{Share}$, resp. $\text{HSS}_2.\text{Share}$) when run on inputs in $\mathcal{R}^{s_1} \times \dots \times \mathcal{R}^{s_N}$ (resp. $\mathcal{R}^{s_1} \times \dots \times \mathcal{R}^{s_{N-1}}$, resp. $\mathcal{R}^{s_1} \times \mathcal{R}^{s_2}$),
- $\text{CC}_N(s_1, \dots, s_N)$ (resp. $\text{CC}_{N-1}(s_1, \dots, s_{N-1})$, resp. $\text{CC}_2(s_1, s_2)$) denotes the size of the smallest boolean circuit computing $\text{HSS}_N.\text{Eval}$ (resp. $\text{HSS}_{N-1}.\text{Eval}$, resp. $\text{HSS}_2.\text{Eval}$) when run on functions of size $(s_1, \dots, s_N)$ (resp. $(s_1, \dots, s_{N-1})$, resp. (s_1, s_2)),

then

$$\begin{cases} S_N(s_1, \dots, s_N) = O(N \cdot S_2(S_{N-1}(s_1, \dots, s_{N-1}), s_N)) \\ \text{CC}_N(s_1, \dots, s_N) = O(N \cdot \text{CC}_2(\text{CC}_{N-1}(s_1, \dots, s_{N-1}), s_N)) \end{cases}$$

Proof. The stated efficiency metrics follow from inspection of the protocol. We focus our efforts on proving correctness and semi-privacy.

- **Correctness.** Let $(x_1, \dots, x_N) \in \mathcal{R}^{n_1} \times \dots \times \mathcal{R}^{n_N}$ (or alternatively let $(x_1, \dots, x_N) \in S^{n_1} \times \dots \times S^{n_N}$ if the classes $\mathcal{C}_2$, $\mathcal{C}_{N-1}$, and $\mathcal{C}_N$ are all parameterised by the same subset $S \subseteq \mathcal{R}$ defining a set of admissible inputs). Consider the probability experiment of $\text{Exp}_{\text{corr}}^{\text{HSS}}(1^\lambda, x_1, \dots, x_N)$:

1. $(\text{share}_i)_{i \in [N]} \xleftarrow{\$} \text{Share}(1^\lambda, (x_1, \dots, x_N))$
2. For $i \in [N]$, $y_i \leftarrow \text{Eval}(i, \text{share}_i, (f_1, \dots, f_N))$
3. Output $y \leftarrow \sum_{i \in [N]} y_i$

Further consider the random variables $(w_i, \tilde{w}_i, z_{i,1}, z_{i,2})_{i \in [N-1]}$ obtained by inlining the definition of Share in the above experiment, identifying the algorithmic variables of Share with random variables defined over the experiment’s probability space. Similarly,

consider the random variables $(y_{N,j})_{j \in [N-1]}$ obtained by inlining the definition of Eval . Consider the following events:

$$(i \in [N-1]) \quad A_i: \quad "y_i + y_{N,i} = g_i(\tilde{w}_i) f_N(x_N)"$$

$$E: \quad "\sum_{i=1}^{N-1} \text{Eval}_{N-1}(i, w_i, (f_1, \dots, f_{N-1})) = f_1(x_1) \dots f_{N-1}(x_{N-1})"$$

Note that each of the A_i ($i \in [N-1]$) holds with all but negligible probability by correctness of $\text{HSS}_{2,i}$ (its input $(\tilde{w}_i, x_N)$ is admissible: $\tilde{w}_i \in \{0_{\mathcal{R}}, 1_{\mathcal{R}}\}^{m_i} \subseteq S^{m_i}$ by the requirement $\{0_{\mathcal{R}}, 1_{\mathcal{R}}\} \subseteq S$, and $x_N \in S^{n_N}$), and that E holds with all but negligible probability by correctness of HSS_{N-1} . Therefore, by a union bound $\Pr[E \cap \bigcap_{i=1}^{N-1} A_i] \geq 1 - \text{negl}(\lambda)$. Observe, via the following equation, that $\Pr[\sum_{i \in [N]} y_i =$

$$f_1(x_1) \dots f_N(x_N) | A_1, \dots, A_{N-1}, E] = 1:$$

$$\begin{aligned} \sum_{i \in [N]} y_i &= \sum_{i \in [N-1]} (y_i + y_{N,i}) \\ &= \sum_{i \in [N-1]} g_i(\tilde{w}_i) f_N(x_N) \quad // \text{ we conditioned on } (A_i)_{i=1}^{N-1} \\ &= \sum_{i \in [N-1]} \text{Eval}_{N-1}(i, w_i, (f_1, \dots, f_{N-1})) f_N(x_N) \\ &\quad // \tilde{w}_i = \text{Encode}_i(w_i) \in \{0_{\mathcal{R}}, 1_{\mathcal{R}}\}^{m_i}, \text{ so by construction of } g_i \\ &\quad // g_i(\tilde{w}_i) = \text{Eval}_{N-1}(i, \text{Decode}_i(\tilde{w}_i), \cdot) = \text{Eval}_{N-1}(i, w_i, \cdot) \\ &= \left(\sum_{i \in [N-1]} \text{Eval}_{N-1}(i, w_i, (f_1, \dots, f_{N-1})) \right) f_N(x_N) \\ &= f_1(x_1) \dots f_N(x_N) \quad // \text{ we conditioned on } E \end{aligned}$$

By the chain rule, $\Pr[\sum_{i \in [N]} y_i = f_1(x_1) \dots f_N(x_N)] \geq 1 - \text{negl}(\lambda)$, hence correctness.

- **Semi-Private Security.** Let us show that the simulator of Fig. 7 satisfies the notion of semi-privacy from Definition 5. Let $\mathcal{Z} \subseteq [N]$ be a set of corrupted parties. If $N \in \mathcal{Z}$, consider the following sequence of hybrids:

1. **Real world:** This is the experiment $\text{Real}_{\mathcal{Z}}^{\text{sp}}$ of Fig. 5. In particular, note that $\text{Real}_{\mathcal{Z}}$ contains the instructions “For $i \in [N-1]$, $(z_{i,1}, z_{i,2}) \xleftarrow{\$} \text{Share}_{2,i}(1^\lambda, (\tilde{w}_i, x_N))$ ”.
2. **Hybrid $\mathcal{H}_{1,i}$ for $i \in [N-1] \cap \mathcal{Z}$:** In this hybrid, the instruction “ $(z_{i,1}, z_{i,2}) \xleftarrow{\$} \text{Share}_{2,i}(1^\lambda, (\tilde{w}_i, x_N))$ ” is replaced with “ $(z_{i,1}, z_{i,2}) \xleftarrow{\$} \text{Sim}_{2,i}(1^\lambda, \mathcal{Z}_i, (m_i, n_N), (\tilde{w}_i, x_N))$ ”. This hybrid is indistinguishable from the previous one by semi-private security of HSS_2 , invoked with respect to corrupted parties $\{i, N\}$.
3. **Hybrid $\mathcal{H}_{2,i}$ for $i \in [N-1] \setminus \mathcal{Z}$:** In this hybrid, the instruction “ $(z_{i,1}, z_{i,2}) \xleftarrow{\$} \text{Share}_{2,i}(1^\lambda, (\tilde{w}_i, x_N))$ ” is replaced with “ $z_{i,2} \xleftarrow{\$} \text{Sim}_{2,i}(1^\lambda, \mathcal{Z}_i, (m_i, n_N), x_N)$ ”. This hybrid is indistinguishable from the previous one by semi-private security of HSS_2 , invoked with respect to corrupted parties $\{N\}$.
4. **Hybrid $\mathcal{H}_3$:** Note that the previous hybrid contains “ $(w_1, \dots, w_{N-1}) \xleftarrow{\$} \text{Share}_{N-1}(1^\lambda, (x_1, \dots, x_{N-1}))$ ”. In this hybrid, it is replaced with “ $(w_j)_{j \in \mathcal{Z}_{<N}} \xleftarrow{\$} \text{Sim}_{N-1}(1^\lambda, \mathcal{Z}_{<N}, (n_i)_{i \in [N-1]}, (x_j)_{j \in \mathcal{Z}_{<N}})$ ”. This hybrid is indistinguishable from the previous one by semi-private security of HSS_{N-1} , invoked with respect to corrupted parties $[N-1] \cap \mathcal{Z}$.

5. **Ideal world:** This is the experiment $\text{Ideal}_{\mathcal{Z}}^{\text{sp}}$ of Fig. 5. It is code-equivalent to hybrid $\mathcal{H}_3$.

If $N \notin \mathcal{Z}$, consider the following sequence of hybrids:

1. **Real world:** This is the experiment $\text{Real}_{\mathcal{Z}}^{\text{sp}}$ of Fig. 5. In particular, note that $\text{Real}_{\mathcal{Z}}$ contains the instructions “For $i \in [N - 1]$, $(z_{i,1}, z_{i,2}) \xleftarrow{\$} \text{Share}_{2,i}(1^\lambda, (\tilde{w}_i, x_N))$ ”.
2. **Hybrid $\mathcal{H}_1$:** In this hybrid, we simply remove all the instructions “ $(z_{i,1}, z_{i,2}) \xleftarrow{\$} \text{Share}_{2,i}(1^\lambda, (\tilde{w}_i, x_N))$ ” for $i \in [N - 1] \setminus \mathcal{Z}$. Since these lines are never used (as neither party i nor party N is corrupted in this case), hybrid $\mathcal{H}_1$ is perfectly indistinguishable from the real world.
3. **Hybrid $\mathcal{H}_{2,i}$ for $i \in \mathcal{Z}$:** The line “ $(z_{i,1}, z_{i,2}) \xleftarrow{\$} \text{Share}_{2,i}(1^\lambda, (\tilde{w}_i, x_N))$ ” is replaced with “ $z_{i,1} \xleftarrow{\$} \text{Sim}_{2,i}(1^\lambda, \mathcal{Z}_i, (m_i, n_N), \tilde{w}_i)$ ”. This hybrid is indistinguishable from the previous one by semi-private security of HSS_2 , invoked with respect to corrupted parties $\{i\}$.
4. **Hybrid $\mathcal{H}_3$:** Note that the previous hybrid contains “ $(w_1, \dots, w_{N-1}) \xleftarrow{\$} \text{Share}_{N-1}(1^\lambda, (x_1, \dots, x_{N-1}))$ ”. In this hybrid, it is replaced with “ $(w_j)_{j \in \mathcal{Z}_{<N}} \xleftarrow{\$} \text{Sim}_{N-1}(1^\lambda, \mathcal{Z}_{<N}, (n_i)_{i \in [N-1]}, (x_j)_{j \in \mathcal{Z}_{<N}})$ ”. This hybrid is indistinguishable from the previous one by semi-private security of HSS_{N-1} , invoked with respect to corrupted parties $[N - 1] \cap \mathcal{Z}$ (i.e. $\mathcal{Z}$).
5. **Ideal world:** This is the experiment $\text{Ideal}_{\mathcal{Z}}^{\text{sp}}$ of Fig. 5. It is code-equivalent to hybrid $\mathcal{H}_3$.

Simulator Sim_N

Requires:

- Sim_N inherits the requirements of HSS_N
- Sim_{N-1} is a simulator for HSS_{N-1} with respect to corrupted parties $\mathcal{Z}_{<N} := \mathcal{Z} \cap [N - 1]$.
- For $i \in [N - 1]$, $\text{Sim}_{2,i}$ is a simulator for HSS_2 with respect to corrupted parties $\mathcal{Z}_i := \mathcal{Z} \cap \{i, N\}$.

Simulation: On input $(1^\lambda, \mathcal{Z}, (n_i)_{i \in [N]}, (x_j)_{j \in \mathcal{Z}})$:

1. $(w_j)_{j \in \mathcal{Z}_{<N}} \xleftarrow{\$} \text{Sim}_{N-1}(1^\lambda, \mathcal{Z}_{<N}, (n_i)_{i \in [N-1]}, (x_j)_{j \in \mathcal{Z}_{<N}})$
2. For $j \in \mathcal{Z}_{<N}$, $\tilde{w}_j \leftarrow \text{Encode}_j(w_j)$
3. If $N \in \mathcal{Z}$:
 - (a) For $i \in \mathcal{Z}_{<N}$:

$$(z_{i,1}, z_{i,2}) \xleftarrow{\$} \text{Sim}_{2,i}(1^\lambda, \mathcal{Z}_i, (m_i, n_N), (\tilde{w}_i, x_N))$$
 - (b) For $i \in [N - 1] \setminus \mathcal{Z}_{<N}$:

$$z_{i,2} \xleftarrow{\$} \text{Sim}_{2,i}(1^\lambda, \mathcal{Z}_i, (m_i, n_N), x_N)$$
 - (c) For $i \in \mathcal{Z}_{<N}$, $\text{share}_i \leftarrow z_{i,1}$
 - (d) $\text{share}_N \leftarrow (z_{i,2})_{i \in [N-1]}$

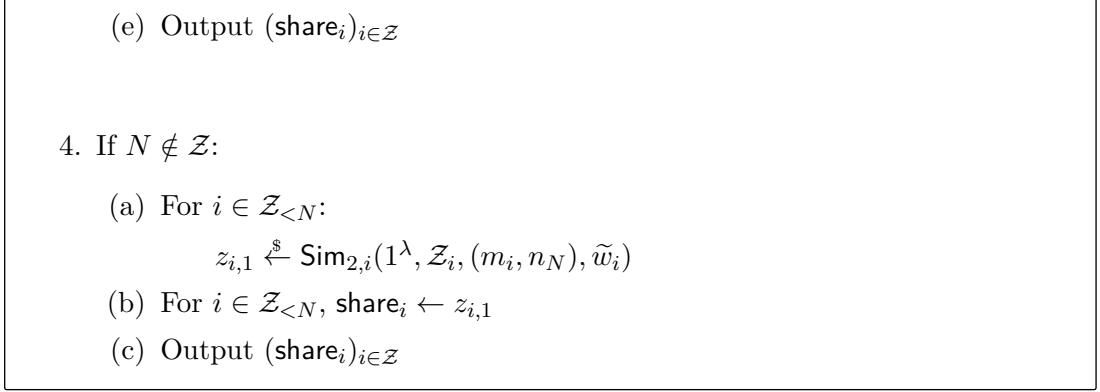


Figure 7: Simulator Sim_N for the construction of Fig. 6.

□

Definition 10 (HSS overhead). Let HSS_2 be a two-party semi-private HSS scheme over some ring $\mathcal{R}$. As in Theorem 9:

- $S_2(s_1, s_2)$ denotes the size of the shares output by $\text{HSS}_2.\text{Share}$ when run on inputs in $\mathcal{R}^{s_1} \times \mathcal{R}^{s_2}$.
- $\text{CC}_2(s_1, s_2)$ denotes the size of the smallest boolean circuit computing $\text{HSS}_2.\text{Eval}$.

We say that HSS_2 has sharing overhead α and computation overhead β (which may depend on the security parameter, but not on the input size or the number of gates in the function) if

$$\begin{aligned} S_2(s_1, s_2) &\leq \alpha \cdot s_1 + O(\text{poly}(\lambda) \cdot s_2) \\ \text{CC}_2(s_1, s_2) &\leq \beta \cdot s_1 + O(\text{poly}(\lambda) \cdot s_2). \end{aligned}$$

Corollary 11 (Multiparty semi-private HSS from semi-private two-party HSS). *Assume the existence of a two-party semi-private HSS (as per Definition 5) HSS_2 for the class $\mathcal{C}_2$ (Definition 8) for every input sizes $n_1, n_2 \in \mathbb{N}$, with sharing overhead $\alpha \geq 1$ and computation overhead $\beta \geq 1$ (Definition 10). Then, for every $N \in \mathbb{N}$ such that $(\max(\alpha, \beta) \cdot N)^N \leq \text{poly}(\lambda)$ (which in particular is satisfied when (1) $N = O(1)$ and $\alpha, \beta \leq \text{poly}(\lambda)$, or (2) when $N \leq \log \lambda / \log \log \lambda$ and $\alpha, \beta \leq \text{polylog}(\lambda)$), there exists a N -party semi-private HSS HSS_N for the class $\mathcal{C}_N$. Moreover,*

- HSS_N outputs shares of size $\text{poly}(\lambda) 2^{O(N)} (\alpha N)^N \sum_{i \in [N]} n_i$;
- $\text{HSS}_N.\text{Eval}$ has computational complexity $\text{poly}(\lambda) 2^{O(N)} (\beta N)^N \sum_{i \in [N]} |f_i|$.

Proof sketch. HSS_N is obtained by applying the construction of Fig. 6 recursively $N-2$ times, starting from HSS_2 ; correctness and semi-privacy follow from Theorem 9 by a straightforward hybrid argument, and the efficiency bounds follow by unrolling the recurrence of Theorem 9. The full proof is given in Appendix C.1. □

4.2 Instantiations from DCR and Class Groups

Recent work on HSS and succinct garbling has constructed semi-private 2-party HSS for B -bounded integers. This was formulated explicitly in [MORS25], who gave a construction from

DCR. While [ILL25a] do not formulate semi-private HSS explicitly, they use it in their CPRF for circuits, which they build from either DCR, or DDH and small exponents over a NIDLS group [ADOS22] (which includes instantiations from class group). In these constructions, every gate being emulated requires $O(1)$ exponentiations and distributed discrete logarithms over a group $\mathbb{G}\mathbb{G}$ large enough to fit both the plaintext (an integer up to B) and some extra $O(K)$ bits for encryption overhead, where K is the key size. Each of these operations can be done with a polynomial-sized circuit.

Theorem 12 ([MORS25, ILL25a]). *Assuming either DCR, or DDH and small exponents over a NIDLS group, there exists a 2-party semi-private HSS for B -bounded integer computation ($\mathcal{R} = \mathbb{Z}$, a set $S \subseteq \mathbb{Z}$ of admissible inputs, and $\mathcal{C}$ the class of B -bounded split functions, i.e. those all of whose wires have absolute value at most B on every input in $S^{n_1} \times S^{n_2}$) with overhead $\alpha = \beta = \text{poly}(K + \log B)$ (Definition 10), where K is the bit size of the secret key.*

In [MORS25, ILL25a] the bound B is stated per input rather than per circuit (an input is admissible for a circuit if no wire exceeds B on it); restricting the class as above is the special case we need, and it is the form of admissibility that Definition 8 expresses, as required by the recursion of Theorem 9. This covers the case where $\mathcal{R}$ contains scalars. However, we'd like to use the extra power of split functions over matrices $\mathcal{R} = \mathbb{Z}^{t \times t}$. This can be built on top of Theorem 12, but one must take care about the efficiency.

Theorem 13. *Assuming either DCR, or DDH and small exponents over a NIDLS group, there exists a 2-party semi-private HSS for B -bounded split functions on an admissible set $S \subseteq \mathbb{Z}^{t \times t}$, as in Theorem 12 but over $\mathcal{R} = \mathbb{Z}^{t \times t}$ instead of $\mathcal{R} = \mathbb{Z}$. It again has overhead $\alpha = \beta = \text{poly}(K + \log B)$, where the sizes of Definition 10 are counted in scalar operations over $\mathbb{Z}$: an input in $\mathbb{Z}^{t \times t}$ counts t^2 , a multiplication of two general matrices counts t^3 , and a multiplication in which one operand has a single nonzero entry counts at most t^2 .*

Proof. The obvious approach is to split each value in the split circuit into t^2 entries, build an arithmetic circuit over $\mathbb{Z}$ out of them, and run the computation inside Theorem 12. However, this introduces an extra factor of t^3 into the overhead, for all the matrix multiplications. Instead, we use an optimising circuit compiler that tries to notice when the matrices are being used to emulate a boolean circuit, as happens when evaluating C_i in Figure 6, which is the bulk of the computation of the construction. In particular, it notices when multiplications by constants force a matrix to have at most 1 nonzero entry, and then just represents the matrix as a scalar from then on. With the encoding of Fig. 6 (one bit per ring element), the semi-private input of C_i consists of matrices with a single entry in $\{0, 1\}$, every wire of the arithmetic emulation of C_i is again such a matrix, and the final recombination $\sum_k \rho_k C_i(X)_k$ multiplies these by constants with a single nonzero entry. This is exactly the situation the compiler detects, so the emulation costs $O(t^2)$ scalar operations per gate and never a t^3 product. $\square$

If we assume standard polynomial security, we must set $K = \text{poly}(\lambda)$, which means polynomial 2-party HSS overhead. If instead we assume subexponential security, then we can set $K = \text{polylog}(\lambda)$ and get significantly lower overhead.

Corollary 14. *Assuming DCR, or DDH and small exponents over a NIDLS group, there is a N -party semi-private HSS for $\mathcal{R} = \mathbb{Z}^{t \times t}$ for B -bounded split functions on an admissible set $S \subseteq \mathbb{Z}^{t \times t}$ (i.e. split functions all of whose wires stay bounded by B on $S^{n_1} \times \dots \times S^{n_N}$), where $B \leq 2^{\text{poly}(\lambda)}$ under polynomial security and $B \leq 2^{\text{polylog} \lambda}$ under subexponential security. When assuming only polynomial security, this holds when $N = O(1)$, and the scheme has computation and key size overhead $\text{poly}(\lambda)^N$. When assuming subexponential security, this holds when $N \leq \log \lambda / \log \log \lambda$, and the scheme has computation and key size overhead*

$\text{poly}(\lambda) \cdot 2^{O(N(\log N + \log \log \lambda))}$, which is $\text{poly}(\lambda)$ in this regime. Both overheads are independent of t when sizes are counted as in Theorem 13. If we instead count the number of matrix gates of the split function, we need to add a multiplicative factor $t^{O(1)}$ (but importantly, not $t^{O(N)}$), since the bound of Corollary 11 is linear in the input and function sizes and the recursion of Fig. 6 only ever emulates boolean circuits. The above also holds when $\mathcal{R}$ is set to a quotient ring of $\mathbb{Z}^{t \times t}$, though the admissible set S and the wire bound B are still understood over $\mathbb{Z}^{t \times t}$, before reduction.

Proof. Plug the two party HSS from Theorem 13 into Corollary 11. In both regimes (namely polynomial security and subexponential security) the bound on B was chosen so that $\log B$ is absorbed by K in the overhead $\alpha = \beta = \text{poly}(K + \log B)$ of Theorem 13, which is therefore $\text{poly}(\lambda)$ in the polynomial regime, and $\text{polylog}(\lambda)$ in the subexponential regime. The recursion itself preserves B -boundedness on S up to a polynomial factor: the emulation circuits g_i of Fig. 6 have all wires in $\{0, 1\}$ except the final recombination, whose value has at most L_i bits, so the split functions handed to HSS_2 at each level are bounded by $\max(B, 2^{L_i})$; since the L_i are polynomial, this is absorbed in the choice of B in both regimes. If $\mathcal{R}$ is a quotient of $\mathbb{Z}^{t \times t}$, have each party quotient its output share locally as a final step. $\square$

As one final extension, we show a way of removing the bound on B , for those parameter regimes of the constructions in the next section (which are very algebraic in nature) in which the numbers get too big for Corollary 14 (see Remark 18).

Corollary 15. *Assuming DCR, or DDH and small exponents over a NIDLS group, there is a N -party semi-private HSS for $\mathcal{R} = \mathbb{Z}^{t \times t}$ (or a quotient ring thereof) for split functions (but with NC^1 instead of RMS for input after the first) with admissible inputs $S = \{0_{\mathcal{R}}, 1_{\mathcal{R}}\}$ (i.e. party i 's input ranges over $\{0_{\mathcal{R}}, 1_{\mathcal{R}}\}^{n_i}$). The assumptions, N limitations, and overheads are as in Corollary 14.*

Proof. Let ℓ be an upper bound on the bit length of all $f_{i,j,k}(x_i)$ (the (j, k) th entry of party i 's matrix). Convert each $f_{i,j,k}(x_i)$ into a function outputting a polynomial $\hat{f}_{i,j,k}(x_i)(t)$, where $\hat{f}_{i,j,k}(x_i)(2) = f_{i,j,k}(x_i)$ and all coefficients of $\hat{f}_{i,j,k}$ are in $\{0, 1\}$. Essentially, the coefficients of $\hat{f}_{i,j,k}(x_i)$ are the bit decomposition of $f_{i,j,k}(x_i)$.⁵ Represent each $\hat{f}_{i,j,k}$ as an $(\ell \cdot N) \times (\ell \cdot N)$ Toeplitz matrix $\hat{F}_{i,j,k}$, such that multiplying the matrices corresponds to multiplying the polynomials. Now put the $\hat{F}_{i,j,k}$ together as blocks of $t \cdot \ell \cdot N \times t \cdot \ell \cdot N$ matrices, and run Corollary 14. Note that this process leads to a B -bounded split function, where $B = O((t \cdot \ell \cdot N)^N)$, i.e. $\log B = O(N \log(t\ell N))$. This is $O(\log \lambda)$ for $N = O(1)$ and $O(\log^2 \lambda / \log \log \lambda)$ for $N \leq \log \lambda / \log \log \lambda$, so in both regimes $B \leq 2^{\text{polylog} \lambda}$ as required by Corollary 14. Finally, take the output shares, convert the Toeplitz submatrices back into polynomials, then evaluate at 2. $\square$

5 From semi-private HSS to HSS

In this section, we show how to upgrade the *semi-private* n -party HSS of Section 4 to *private* n -party HSS (Definition 4), at the cost of restricting the function class. We capture functions with polynomially bounded degree d and mixed Waring rank r (Definition 6), i.e. functions of the form $f = \sum_{j=1}^r \alpha_j f_j^{d_j}$ with the f_j affine and $d_j \leq d$. The idea is to express $f(x^{(1)} + \dots + x^{(n)})$, where the $(x^{(i)})_{i \in [n]}$ are additive shares of the input, as a split function (Definition 8) of the shares. We provide two possible ways of doing so, each one optimising for a different parameter:

⁵This bit decomposition is why we want the functions f_i to be defined using NC^1 instead of as RMS programs.

- *Any degree, restricted number of parties.* In Section 5.1, we use a single n -party semi-private HSS for $\mathcal{C}_n$ over the matrix ring $\mathcal{R}^{(d+1) \times (d+1)}$: each $f_j(x)^{d_j}$ is recovered as an entry of a product of n matrices, one per share. This handles any polynomial degree d , but requires n -party semi-private HSS, and hence inherits the restriction on n of Corollary 14 (Theorem 16, Remark 18).
- *Any number of parties, restricted degree.* In Section 5.2, we instead use k -party semi-private HSS for $\mathcal{C}_k$ over $\mathcal{R}$ itself, for every “committee” of $k \in [2, \min(d, n)]$ parties. By the multinomial theorem, every monomial of $(\sum_i h_i(x^{(i)}))^{d_j}$ involves at most d_j of the shares, and is computed by the committee of the parties it involves. This supports any polynomial number of parties n , but only degree $d = O(\log \lambda / \log n)$, since the parties must hold shares for all $\sim n^d$ committees (Theorem 19).

Finally, in Section 5.3, we observe (following [BMO⁺25]) that point functions have polynomially bounded degree and mixed Waring rank, so that either construction yields multiparty *distributed point functions*.

5.1 From semi-private HSS for $\mathcal{C}_d$ to private HSS for functions of polynomially bounded degree and (mixed) Waring rank

HSS

// n -party HSS (as per Definition 4) for the class of degree- d m -variate functions with polynomially bounded Waring rank.

Requires:

- $\text{spHSS} = (\text{Share}^{\text{sp}}, \text{Eval}^{\text{sp}})$ is an n -party semi-private HSS (Definition 5) for the class $\mathcal{C}_n$ (Definition 8) as parameterised by (referring to the notations of Definition 8) $\ell \leftarrow n$, $\mathcal{R} \leftarrow \mathcal{R}^{(d+1) \times (d+1)}$, $n_1, \dots, n_\ell \leftarrow m$, and admissible inputs $S \leftarrow \text{scalar-to-matrix}(\mathcal{R}) := \{\text{scalar-to-matrix}(x) : x \in \mathcal{R}\}$. That is, each function in $\mathcal{C}_n$ takes as input an element of $\underbrace{(\mathcal{R}^{(d+1) \times (d+1)})^m \times \dots \times (\mathcal{R}^{(d+1) \times (d+1)})^m}_{n \text{ times}}$ and outputs an element of $\mathcal{R}^{(d+1) \times (d+1)}$, and its admissible inputs are those all of whose matrices lie in $\text{scalar-to-matrix}(\mathcal{R})$.
- $\text{scalar-to-matrix} : \mathcal{R} \rightarrow \mathcal{R}^{(d+1) \times (d+1)}$ maps a scalar x to the $(d+1)$ -by- $(d+1)$ matrix with x in position $(0, 0)$, and 0 elsewhere.

$\text{HSS.Share}(1^\lambda, x = (x_1, \dots, x_m)) :$

1. For $i \in [2, n]$, $x^{(i)} = (x_1^{(i)}, \dots, x_m^{(i)}) \xleftarrow{\$} \mathcal{R}^m$
2. $x^{(1)} \leftarrow x - \sum_{i=2}^n x^{(i)}$
3. For $i \in [n]$, $X_i \leftarrow (\text{scalar-to-matrix}(x_j^{(i)}))_{j \in [m]}$
4. $(\text{share}_1, \dots, \text{share}_n) \xleftarrow{\$} \text{spHSS.Share}^{\text{sp}}(1^\lambda, X_1, \dots, X_n)$
5. Output $(\text{share}_1, \dots, \text{share}_n)$

$\text{HSS.Eval}(i, \text{share}_i, f) :$

1. Parse the description of f so as to retrieve $(f_1, \dots, f_r)$, $(\alpha_1, \dots, \alpha_r)$, and $(d_1, \dots, d_r)$ such that $f = \sum_{j=1}^r \alpha_j f_j^{d_j}$ (c.f. Definition 6).

2. For $j = 1, \dots, r$:

(a) $W_{i,j} \leftarrow \text{spHSS.Eval}^{\text{sp}}(i, \text{share}_i, (g_{j,1}, \dots, g_{j,n}))$,
where the $(g_{j,\ell}: (\mathcal{R}^{(d+1) \times (d+1)})^m \rightarrow \mathcal{R}^{(d+1) \times (d+1)})_{\ell \in [n]}$ are defined as:

$g_{j,\ell}(Z = (Z_1, \dots, Z_m))$:

1. $z \leftarrow (Z_1[0][0], \dots, Z_m[0][0])$

2. $a \leftarrow \begin{cases} f_j(z) - f_j(0_{\mathcal{R}^m}) & \text{if } \ell \in [2, n] \\ f_j(z) & \text{if } \ell = 1 \end{cases}$

3. Output $\left(\binom{j}{i} a^{j-i} \right)_{(i,j) \in [0,d]^2}$

// $W_{i,j} = \langle g_{j,1}(X_1) \dots g_{j,n}(X_n) \rangle_{[n]}^{(i)}$

(b) $w_{i,j} \leftarrow \underbrace{(1, 0, \dots, 0)}_{d+1} \cdot W_{i,j} \cdot \underbrace{(0, \dots, 0, 1, 0, \dots, 0)}_{d+1}^{\overbrace{d_j} \quad \overbrace{d-d_j}}^\top // \langle (f_j(x))^{d_j} \rangle_{[n]}^{(i)}$

(c) $y_{i,j} \leftarrow \alpha_j w_{i,j} // \langle \alpha_j (f_j(x))^{d_j} \rangle_{[n]}^{(i)} \leftarrow \alpha_j \langle (f_j(x))^{d_j} \rangle_{[n]}^{(i)}$

3. $y_i \leftarrow \sum_{j=1}^r y_{i,j} // \langle f(x) \rangle_{[n]}^{(i)} \leftarrow \sum_{j=1}^r \langle \alpha_j (f_j(x))^{d_j} \rangle_{[n]}^{(i)}$

4. Output y_i

Figure 8: n -HSS for functions of polynomially bounded degree and mixed Waring rank, from n -spHSS for $\mathcal{C}_n$.

Theorem 16. *Let $\mathcal{R}(\lambda)$ be a finite commutative ring. If its requirements are met, then the construction of Fig. 8 is an n -party HSS (as per Definition 4) for the class of all functions $\mathcal{R}^m \rightarrow \mathcal{R}$ with degree d and mixed-degree Waring rank (Definition 6) at most r .*

Proof. • **Correctness.** We start by establishing a useful lemma. For $a \in \mathcal{R}$, define $M_a := \left(\binom{j}{i} a^{j-i} \right)_{(i,j) \in [0,d]^2}$ (with the convention that $\binom{j}{i} a^{j-i} = 0$ if $i > j$).

Lemma 17. *For all $(a_1, \dots, a_n) \in \mathcal{R}^n$ and for all $d' \in [0, d]$,*

$$\underbrace{(1, 0, \dots, 0)}_{d+1} \cdot M_{a_1} \cdots M_{a_n} \cdot \underbrace{(0, \dots, 0, 1, 0, \dots, 0)}_{d+1}^{\overbrace{d'} \quad \overbrace{d-d'}}^\top = (a_1 + \dots + a_n)^{d'}$$

Lemma 17 follows from a direct application of the binomial theorem, and is therefore deferred to Appendix C.2.

We now resume the proof of correctness. Recall that $X_i = (\text{scalar-to-matrix}(x_k^{(i)}))_{k \in [m]}$, so that step 1 of $g_{j,i}(X_i)$ recovers $z = x^{(i)}$. By correctness of **spHSS**, the following equation holds with all but negligible probability

$$\begin{aligned} \sum_{i=1}^n W_{i,j} &= g_{j,1}(X_1) \cdots g_{j,n}(X_n) \\ &= M_{f_j(x^{(1)})} \cdot M_{f_j(x^{(2)}) - f_j(0_{\mathcal{R}^m})} \cdots M_{f_j(x^{(n)}) - f_j(0_{\mathcal{R}^m})} \end{aligned} \quad (1)$$

Conditioned on the above equation holding, note that

$$\begin{aligned} \sum_{i=1}^n w_{i,j} &= \sum_{i=1}^n (1, 0, \dots, 0) \cdot W_{i,j} \cdot (0, \dots, 0, 1, 0, \dots, 0) \\ &= (1, 0, \dots, 0) \cdot (\sum_{i=1}^n W_{i,j}) \cdot (0, \dots, 0, 1, 0, \dots, 0) \\ &= (f_j(x^{(1)}) + f_j(x^{(2)}) - f_j(0_{\mathcal{R}^m}) + \cdots + f_j(x^{(n)}) - f_j(0_{\mathcal{R}^m}))^{d_j} \end{aligned}$$

where the last equation is obtained by applying Lemma 17. Because $f_j: \mathcal{R}^m \rightarrow \mathcal{R}$ is affine, $f_j - f_j(0_{\mathcal{R}^m})$ is linear. Therefore $\sum_{i=1}^n (f_j(x^{(i)}) - f_j(0_{\mathcal{R}^m})) = f_j(\sum_{i=1}^n x^{(i)}) - f_j(0_{\mathcal{R}^m})$. It follows that $f_j(x^{(1)}) + \sum_{i=2}^n (f_j(x^{(i)}) - f_j(0_{\mathcal{R}^m})) = f_j(\sum_{i=1}^n x^{(i)}) = f_j(x)$. In turn, still conditioning on Eq. (1) holding:

$$\begin{aligned} \sum_{i=1}^n w_{i,j} &= (f_j(x^{(1)}) + f_j(x^{(2)}) - f_j(0_{\mathcal{R}^m}) + \cdots + f_j(x^{(n)}) - f_j(0_{\mathcal{R}^m}))^{d_j} \\ &= (f_j(\sum_{i=1}^n x^{(i)}))^{d_j} \\ &= (f_j(x))^{d_j} \end{aligned}$$

And finally

$$y = \sum_{i=1}^n y_i = \sum_{i=1}^n \sum_{j=1}^r y_{i,j} = \sum_{j=1}^r \alpha_j \sum_{i=1}^n w_{i,j} = \sum_{j=1}^r \alpha_j (f_j(x))^{d_j} = f(x) .$$

- **Security.** Let Sim^{sp} be a simulator (as per Fig. 5) for **spHSS**. Let $\mathcal{Z} \subsetneq [n]$ be the set of corrupted parties (recall that Definition 4 only quantifies over $\mathcal{Z} \subsetneq [n]$, *i.e.* there is at least one honest party). Then the $(x^{(i)})_{i \in \mathcal{Z}}$ are (in the real world) additive shares of $x \in \mathcal{R}^m$ which hold no information about it. This means that the fact the $(X_i)_{i \in \mathcal{Z}}$ are *semi-private* inputs to **spHSS** does not jeopardise x . The simulator can therefore sample $(x^{(i)})_{i \in \mathcal{Z}} \xleftarrow{\$} (\mathcal{R}^m)^{\mathcal{Z}}$, then set $X_i \leftarrow (\text{scalar-to-matrix}(x_k^{(i)}))_{k \in [m]}$ for $i \in \mathcal{Z}$, and finally run $(\text{share}_i)_{i \in \mathcal{Z}} \xleftarrow{\$} \text{Sim}^{\text{sp}}(1^\lambda, \mathcal{Z}, (m, \dots, m), (X_i)_{i \in \mathcal{Z}})$. □

Remark 18 (Instantiating the requirements of Fig. 8). Corollary 14 yields **spHSS** over a quotient ring of $\mathbb{Z}^{t \times t}$, but only for B -bounded computations, where B bounds every wire of the computation lifted to $\mathbb{Z}^{t \times t}$. We remark that Fig. 8 can meet this requirement for $\mathcal{R} = \mathbb{Z}_p$ (with $p > d$ prime, which is needed for the DPF application below) and $t = d + 1$, so that $\mathcal{R}^{(d+1) \times (d+1)}$ is the quotient of $\mathbb{Z}^{(d+1) \times (d+1)}$ by p . The coordinates of the $x^{(i)}$ and the coefficients of the affine functions f_j are elements of $\mathbb{Z}_p$, which we view as elements of $[0, p)$. We in turn define the set of admissible inputs as matrices of the form $\text{scalar-to-matrix}(x)$ for $x \in [0, p)$. Each $g_{j,\ell}$ is then evaluated over $\mathbb{Z}$. Note that the scalar a of step 2 is bounded by $(m + 1)p^2$, and so the coefficients of M_a (the $\binom{v}{u} a^{v-u}$) are bounded by $2^d ((m + 1)p^2)^d$. Multiplying at most n of these $(d + 1) \times (d + 1)$ matrices yields a matrix whose coefficients are at most $(d + 1)^{n-1} (2^d ((m + 1)p^2)^d)^n$. Therefore all wires are bounded by B such that

$$\log B = O(n d \log(mp)) .$$

Thus the $g_{j,\ell}$ are B -bounded split functions on $S = \text{scalar-to-matrix}([0, p))$ in the sense of Corollary 14. Since $\sum_{i \in [n]} x^{(i)} \equiv x \pmod{p}$ and f has integer coefficients, reducing the

output shares modulo p (as in the proof of Corollary 14) yields additive shares of $f(x)$ over $\mathbb{Z}_p$. In the polynomial hardness regime (*i.e.* $n = O(1)$ and $d, m, \log p \leq \text{poly}(\lambda)$) this gives $B \leq 2^{\text{poly}(\lambda)}$, which Corollary 14 allows in that regime. In the subexponential hardness regime, the corollary requires $\log B \leq \text{polylog}(\lambda)$, hence $d \leq \text{polylog}(\lambda)$. Here, we use the bit-decomposition variant of Corollary 15 (the inputs are now the bits of the $x_j^{(i)}$, and the entries of M_a are computed in NC^1 ⁶) to support larger degrees. The same discussion applies to Fig. 9, except that n replaced by the committee size $|S| \leq d$.

5.2 An alternative, committee-based, construction

In this section, we provide an alternative way to build n -party HSS for m -variate functions with mixed Waring rank r and degree d . As in Section 5.1, the starting point is the observation that if the input $x \in \mathcal{R}^m$ is additively secret-shared as $x = x^{(1)} + \dots + x^{(n)}$ with $x^{(i)} \in \mathcal{R}^m$,

$$\begin{aligned} f(x) &= \sum_{j=1}^r \alpha_j \cdot (f_j(x^{(1)} + \dots + x^{(n)}))^{d_j} \\ &= \sum_{j=1}^r \alpha_j \cdot \left(\sum_{i=1}^{n-1} [f_j(x^{(i)}) - f_j(0_{\mathcal{R}^m})] + f_j(x^{(n)}) \right)^{d_j} \end{aligned}$$

If we denote $h_{i,j}(x^{(i)}) \leftarrow f_j(x^{(i)}) - (1 - \delta_{i,n})f_j(0_{\mathcal{R}^m})$, the core problem is therefore to allow the n parties to generate shares of $(h_{1,j}(x^{(1)}) + \dots + h_{n,j}(x^{(n)}))^{d_j}$, where $h_{i,j}(x^{(i)})$ need not be kept private from party i . The core observation of this section is that each of this polynomial's $\sim n^{d_j}$ monomials depends on at most d_j of the $x^{(i)}$. So if party i knows $x^{(i)}$, and the parties hold semi-private HSS shares of $(x^{(i)})_{i \in S}$ for *every* subset S of at least two and at most d parties, the parties can generate shares of each of these monomials (and therefore, by linear combinations, of $f(x)$): the monomials involving a single $x^{(i)}$ are computed locally by party i , the others by the committee S of the parties they involve.

HSS

// n -party HSS (as per Definition 4) for the class of degree- d m -variate functions with polynomially bounded mixed Waring rank – committee-based construction.

Requires:

- For each $k \in [2, \min(d, n)]$, $\text{spHSS}_k = (\text{Share}_k^{\text{sp}}, \text{Eval}_k^{\text{sp}})$ is a k -party semi-private HSS (Definition 5) for the class $\mathcal{C}_k$ (Definition 8) over $\mathcal{R}$, as parameterised by input sizes $n_1, \dots, n_k \leftarrow m$.
- For $S \subseteq [n]$ with elements $s_1 < \dots < s_{|S|}$ and $i \in S$, $\text{rank}_S(i) \in [|S|]$ denotes the rank of i in S (*i.e.* $s_{\text{rank}_S(i)} = i$).
- For $i \in [n]$ and an affine $f_j: \mathcal{R}^m \rightarrow \mathcal{R}$, $h_{i,j}: \mathcal{R}^m \rightarrow \mathcal{R}$ denotes $z \mapsto f_j(z) - f_j(0_{\mathcal{R}^m})$ if $i \in [n-1]$, and $h_{n,j} := f_j$. // $h_{i,j}$ is linear for $i < n$, so $\sum_{i \in [n]} h_{i,j}(x^{(i)}) = f_j(\sum_i x^{(i)})$.

$\text{HSS.Share}(1^\lambda, x = (x_1, \dots, x_m)) :$

1. For $i \in [2, n]$, $x^{(i)} \xleftarrow{\$} \mathcal{R}^m$
2. $x^{(1)} \leftarrow x - \sum_{i=2}^n x^{(i)}$

⁶Note that an iterated product of at most d integers of $O(\log(mp))$ bits is in $\text{TC}^0 \subseteq \text{NC}^1$.

3. For each $S \subseteq [n]$ with $2 \leq |S| \leq \min(d, n)$, $(\text{share}_{i,S})_{i \in S} \xleftarrow{s} \text{spHSS}_{|S|}.\text{Share}^{\text{sp}}(1^\lambda, (x^{(i)})_{i \in S})$ // $x^{(i)}$ is the semi-private input of party $\text{rank}_S(i)$ of $\text{spHSS}_{|S|}$.
4. For $i \in [n]$, set $\text{share}_i \leftarrow (x^{(i)}, (\text{share}_{i,S})_{S \ni i})$ // Party i receives $x^{(i)}$ in the clear; it is semi-private to i in every committee anyway.
5. Output $(\text{share}_1, \dots, \text{share}_n)$

$\text{HSS.Eval}(i, \text{share}_i, f)$:

1. Parse the description of f so as to retrieve $(f_1, \dots, f_r)$, $(\alpha_1, \dots, \alpha_r)$, and $(d_1, \dots, d_r)$ such that $f = \sum_{j=1}^r \alpha_j f_j^{d_j}$ (c.f. Definition 6).
2. Parse share_i as $(x^{(i)}, (\text{share}_{i,S})_{S \ni i})$
3. For $j = 1, \dots, r$: // Compute $\langle (f_j(x))^{d_j} \rangle_{[n]}^{(i)}$
 - (a) If $d_j = 0$: $w_{i,j} \leftarrow 1$ if $i = 1$, and $w_{i,j} \leftarrow 0$ otherwise. // $f_j^0 = 1$.
 - (b) Otherwise:

$$w_{i,j} \leftarrow h_{i,j}(x^{(i)})^{d_j} + \sum_{\substack{S \ni i \\ 2 \leq |S| \leq d_j}} \sum_{\substack{\vec{k} \in [1, d_j]^{|S|} \\ \|\vec{k}\|_1 = d_j}} \binom{d_j}{k_1, \dots, k_{|S|}}.$$

$$\text{spHSS}_{|S|}.\text{Eval}^{\text{sp}}(\text{rank}_S(i), \text{share}_{i,S}, (g_{S,j,\vec{k},1}, \dots, g_{S,j,\vec{k},|S|}))$$

where, for $S = \{s_1 < \dots < s_{|S|}\}$ and $\ell \in [|S|]$, $g_{S,j,\vec{k},\ell}: \mathcal{R}^m \rightarrow \mathcal{R}$ is the arithmetic circuit computing $z \mapsto h_{s_\ell,j}(z)^{k_\ell}$.

// The first term is the $S = \{i\}$ term of the multinomial expansion of $(\sum_{i' \in [n]} h_{i',j}(x^{(i')}))^{d_j}$; party i computes it locally since it holds $x^{(i)}$. The committees of size two or more supply the mixed monomials. Powers of an affine function of the input are computable by RMS programs, so $(g_{S,j,\vec{k},\ell})_{\ell \in [|S|]} \in \mathcal{C}_{|S|}$.

- (c) $y_{i,j} \leftarrow \alpha_j w_{i,j}$ // $\langle \alpha_j (f_j(x))^{d_j} \rangle_{[n]}^{(i)} \leftarrow \alpha_j \langle (f_j(x))^{d_j} \rangle_{[n]}^{(i)}$
4. $y_i \leftarrow \sum_{j=1}^r y_{i,j}$ // $\langle f(x) \rangle_{[n]}^{(i)} \leftarrow \sum_{j=1}^r \langle \alpha_j (f_j(x))^{d_j} \rangle_{[n]}^{(i)}$
5. Output y_i

Figure 9: n -HSS for functions of polynomially bounded degree and mixed Waring rank, from k -spHSS for $\mathcal{C}_k$, $k \in [2, \min(d, n)]$ – committee-based construction.

Theorem 19. *Let $\mathcal{R}(\lambda)$ be a finite commutative ring. If its requirements are met and if $n^d, r, d^d \leq \text{poly}(\lambda)$, then the construction of Fig. 9 is an n -party HSS (as per Definition 4) for the class of all functions $\mathcal{R}^m \rightarrow \mathcal{R}$ with degree d and mixed-degree Waring rank (Definition 6) at most r .*

Proof sketch. The proof follows the same lines as that of Theorem 16: correctness follows

from the multinomial theorem (grouping the monomials of $(y_1 + \dots + y_n)^{d_j}$ by their set S of non-zero variables) together with a union bound over the polynomially many committees, and security from a hybrid argument replacing the sharing of each committee S with $S \cap \mathcal{Z} \neq \emptyset$ by its semi-private simulator. The full proof is given in Appendix C.3. $\square$

5.3 Applications to multiparty Distributed Point Functions

Theorem 20. *Let $n \leq \text{poly}(\lambda)$ and $\mathcal{R}$ be a field of characteristic 0 or greater than d . Assuming there exists ℓ -party HSS (Definition 4) for degree- d functions with polynomially bounded mixed-degree Waring rank (Definition 6) over $\mathcal{R}$, there exist ℓ -party distributed point functions over $\mathcal{R}$ with domain $[n]^{d-1}$.*

Proof. A distributed point function over $\mathcal{R}$ is defined as a function secret-sharing scheme for the class of point functions $\left\{ \begin{array}{ccc} f_{\alpha, \beta}: [n]^{d-1} & \rightarrow & \mathcal{R} \\ x & \mapsto & \beta \cdot (x == \alpha) \end{array} \right\}_{\alpha \in [n]^{d-1}, \beta \in \mathcal{R}}$.

By duality between function secret-sharing and homomorphic secret sharing, it suffices to build a homomorphic secret-sharing scheme for the class $\left\{ \begin{array}{ccc} f_{\cdot, \cdot}(x): [n]^{d-1} \times \mathcal{R} & \rightarrow & \mathcal{R} \\ (\alpha, \beta) & \mapsto & \beta \cdot (x == \alpha) \end{array} \right\}_{x \in [n]^{d-1}}$. We convert each position of the

vector α into 1-hot representation, by creating a set of n vectors $\alpha_i \in \{0, 1\}^{d-1}$ where $\alpha_{j,i} = 1$ if and only if $\alpha_i = j$. As observed in (the proof of) [BMO⁺25, Corollary 9], for each $x \in [n]^{d-1}$ the function $f_{\cdot, \cdot}(x): ((\alpha_1, \dots, \alpha_n), \beta) \mapsto f_{\alpha, \beta}(x)$ can be computed by a function with polynomially bounded degree and polynomially bounded mixed Waring rank. Indeed:

$$\begin{aligned} \beta \cdot (x == \alpha) &= \beta \left[\left(\sum_{i=1}^{d-1} (x_i == \alpha_i) \right) == d - 1 \right] \\ &= \beta [A_x(\alpha) == d - 1] && \text{where } A_x: [n]^{d-1} \rightarrow \mathbb{Z} \\ &&& (\alpha_i)_{i \in [d-1]} \mapsto \sum_{i=1}^{d-1} \mathbf{1}_{x_i, \alpha_i} \\ &= \frac{\beta}{(d-1)!} \prod_{i=0}^{d-2} (A_x(\alpha) - i) \\ &= \frac{1}{(d-1)!} \left[\sum_{i=0}^{d-1} \beta c_i \cdot A_x(\alpha)^i \right] \\ &= \sum_{i=0}^{d-1} \frac{c_i}{(d-1)!} \cdot \beta \cdot A_x(\alpha)^i \end{aligned}$$

where the $(c_i)_{i=0}^{d-1}$ are the (polynomial-time computable) coefficients of the degree- $(d-1)$ polynomial $\prod_{i=0}^{d-2} (X - i)$. This is almost in polynomially-bounded Waring rank form, except for the multiplication by β (which raises the degree to d). This can be handled using [BMO⁺25, Lemma 11].⁷ $\square$

AI Disclosure.

No LLM was used at any point during the conception phase of this paper, and the original manuscript was fully written without AI assistance. Anthropic’s Fable 5 was used as an after-the-fact external reviewer to help find minor typos and inconsistent notations. Additionally,

⁷The Lemma is stated for functions over $\mathbb{Q}$, but the only feature of $\mathbb{Q}$ it uses is being able to divide by integers up to d , which are invertible in $\mathcal{R}$ by the assumption on its characteristic (Remark 7).

it was used to transcribe Fig. 1 from a hand-drawn whiteboard figure to a faithful tikz copy.

Acknowledgments

This research was supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme under grant agreement number 101124977 (DECRYPSIS), the Danish Independent Research Council under Grant-IDs DFF-3103-00077B (CryptoDigi) and DFF-0165-00107B (C3PO), and by a grant from the STIBO foundation.

Part of this work was conducted while P. Meyer, L. Roy, and P. Scholl were visiting the Simons Institute for the Theory of Computing.

References

- [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 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452. Springer, Cham, August 2022.
- [ARS24] Damiano Abram, Lawrence Roy, and Peter Scholl. Succinct homomorphic secret sharing. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 301–330. Springer, Cham, May 2024.
- [BCM23] Elette Boyle, Geoffroy Couteau, and Pierre Meyer. Sublinear-communication secure multiparty computation does not require FHE. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 159–189. Springer, Cham, April 2023.
- [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, Berlin, Heidelberg, April 2015.
- [BGI16a] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Berlin, Heidelberg, August 2016.
- [BGI16b] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.
- [BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193. Springer, Cham, April / May 2017.
- [BGIK22] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 121–151. Springer, Cham, August 2022.

- [BMO⁺25] Amik Raj Behera, Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Privately constrained PRFs from DCR: Puncturing and bounded waring rank. In Benny Applebaum and Huijia (Rachel) Lin, editors, *TCC 2025, Part I*, volume 16268 of *LNCS*, pages 301–332. Springer, Cham, December 2025.
- [CBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE Computer Society Press, May 2015.
- [CKY25] Geoffroy Couteau, Naman Kumar, and Xiayi Ye. Multiparty homomorphic secret sharing and more from LPN and MQ. In *TCC 2025*, 2025.
- [CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 194–224. Springer, Cham, April 2023.
- [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, *SCN 22*, volume 13409 of *LNCS*, pages 540–563. Springer, Cham, September 2022.
- [COSL24] Ilaria Chillotti, Emmanuela Orsini, Peter Scholl, and Barry Van Leeuwen. Scooby: Improved multi-party homomorphic secret sharing based on FHE. *Inf. Comput.*, 297:105133, 2024.
- [DGI⁺19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Cham, August 2019.
- [DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Berlin, Heidelberg, August 2016.
- [DIJL23] Quang Dao, Yuval Ishai, Aayush Jain, and Huijia Lin. Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 315–348. Springer, Cham, August 2023.
- [DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: Scaling private contact discovery. *PoPETs*, 2018(4):159–178, October 2018.
- [Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.
- [GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, Berlin, Heidelberg, May 2014.
- [ILL25a] Yuval Ishai, Hanjun Li, and Huijia Lin. Succinct homomorphic MACs from groups and applications. In *66th FOCS*, pages 2505–2545. IEEE Computer Society Press, December 2025.

- [ILL25b] Yuval Ishai, Hanjun Li, and Huijia Lin. A unified framework for succinct garbling from homomorphic secret sharing. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 390–425. Springer, Cham, August 2025.
- [MORS25] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Silent circuit relinearisation: Sublinear-size (boolean and arithmetic) garbled circuits from DCR. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 426–458. Springer, Cham, August 2025.
- [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, Cham, October 2021.
- [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, Cham.
- [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.

A Defining (semi-private) “high-end” HSS

A.1 (Private) “high-end” HSS.

The notion of “high-end” HSS differs from the notion from Section 3.1.1. Firstly, we allow for inputs to be shared one-by-one instead of all together. Secondly, it supports the notion of “offline-online” sharing, which allows us to sample all but one of the parties’ input shares before having to commit to the input being shared.

Definition 21 (“High-end” HSS). Let $\ell \in \mathbb{N} \setminus \{0, 1\}$. Let $\mathcal{R} = \mathcal{R}(\lambda)$ be a ring and $n \in \text{poly}(\lambda)$. An ℓ -party additive HSS scheme supporting a class of computations $\mathcal{C} \subseteq \mathcal{R}^n \rightarrow \mathcal{R}$ is a triple of polynomial-time algorithms $\text{HSS} = (\text{Setup}, \text{Share}, \text{Eval})$, where **Setup** and **Share** are probabilistic but **Eval** is deterministic, with the following syntax:

- $\text{HSS.Setup}(1^\lambda)$: On input the security parameter 1^λ , the setup algorithm **Setup** outputs a secret key sk .
- $\text{HSS.Share}(1^\lambda, \text{sk}, y)$: On input the security parameter 1^λ , a secret key sk , and an input $y \in \mathcal{R}$, the input sharing algorithm **Share** outputs an ℓ -tuple of input shares $(y^{(1)}, \dots, y^{(\ell)})$.
- $\text{HSS.Eval}(j, (x_1^{(j)}, \dots, x_n^{(j)}), C)$: On input a party index j , the j^{th} shares $(x_i^{(j)})_{i \in [n]}$ of the n inputs (each of which was shared by a separate call to **Share**), and a supported evaluation circuit $C \in \mathcal{C}$, the homomorphic evaluation algorithm **Eval** returns the j^{th} output share $y^{(j)} \in \mathcal{R}$.

We additionally require that HSS satisfy the following notions of correctness and security.

- **Correctness.** For any sufficiently large $\lambda \in \mathbb{N}$, for any circuit $C \in \mathcal{C}$, any C -admissible inputs $(x_1, \dots, x_n) \in \mathcal{R}^n$,

$$\Pr \left[y = C(x_1, \dots, x_n) : y \xleftarrow{\$} \text{Exp}_{\text{corr}}^{\text{HSS}}(1^\lambda, x_1, \dots, x_n) \right] \geq 1 - \text{negl}(\lambda)$$

where $\text{Exp}_{\text{corr}}^{\text{HSS}}$ is defined in Fig. 10.

- **Security.** There exists a *p.p.t.* algorithm **Sim** (a simulator) such that, for every set of corrupted parties $\mathcal{Z} \subsetneq [\ell]$, every $q \in \text{poly}(\lambda)$, and any $x_1, \dots, x_q \in \mathcal{R}$, the outputs of the experiments $\text{Real}_{\mathcal{Z}}(1^\lambda, x_1, \dots, x_q)$ and $\text{Ideal}_{\mathcal{Z}}(1^\lambda, q)$ (Fig. 11) are computationally indistinguishable.⁸

If HSS satisfies the following *optional* property, we say that HSS is *offline-online*.

- **Offline-online sharing.** We say the input-sharing algorithm **Share** is *offline-online* if there exist *p.p.t.* algorithms $\text{Share}_1, \dots, \text{Share}_\ell$ with the following syntax and properties:

⁸A single secret key is used to share all the inputs of the correctness experiment (Fig. 10), which is why security is required to hold for polynomially many inputs shared under the same key. The case $q = 1$ is the usual single-input notion.

- For $i \in [2, \ell]$, $\text{Share}_i(1^\lambda, \text{sk}, \text{aux}_\ell, \dots, \text{aux}_{i+1})$: On input the security parameter 1^λ , the secret key sk , and the auxiliary information $(\text{aux}_j)_{j \in [i+1, \ell]}$ output by the input-sharing algorithms of the parties with a larger index (for $i = \ell$, there is none), the i^{th} input-sharing algorithm Share_i produces an i^{th} input share $x^{(i)}$ and auxiliary information aux_i . The offline algorithms are thus run in the order $\text{Share}_\ell, \text{Share}_{\ell-1}, \dots, \text{Share}_2$ (the same order as in Definition 22).
- $\text{Share}_1(1^\lambda, \text{sk}, \text{aux}_2, \dots, \text{aux}_\ell, x)$: On input the security parameter 1^λ , the secret key sk , auxiliary information $(\text{aux}_i)_{i \in [2, \ell]}$, and an input x , the first input-sharing algorithm Share_1 produces a first input share $x^{(1)}$.
- **Offline-online correctness.** For any sufficiently large $\lambda \in \mathbb{N}$, for any sk in the support of $\text{Setup}(1^\lambda)$, and for any $x \in \mathcal{R}(\lambda)$, the following distributions are identically distributed:

$$\left\{ (x^{(1)}, \dots, x^{(\ell)}) : (x^{(1)}, \dots, x^{(\ell)}) \stackrel{\$}{\leftarrow} \text{Share}(1^\lambda, \text{sk}, x) \right\}$$

and

$$\left\{ (x^{(1)}, \dots, x^{(\ell)}) : \begin{array}{ll} (x^{(\ell)}, \text{aux}_\ell) & \stackrel{\$}{\leftarrow} \text{Share}_\ell(1^\lambda, \text{sk}) \\ (x^{(\ell-1)}, \text{aux}_{\ell-1}) & \stackrel{\$}{\leftarrow} \text{Share}_{\ell-1}(1^\lambda, \text{sk}, \text{aux}_\ell) \\ & \dots \\ (x^{(2)}, \text{aux}_2) & \stackrel{\$}{\leftarrow} \text{Share}_2(1^\lambda, \text{sk}, \text{aux}_\ell, \dots, \text{aux}_3) \\ x^{(1)} & \stackrel{\$}{\leftarrow} \text{Share}_1(1^\lambda, \text{sk}, \text{aux}_2, \dots, \text{aux}_\ell, x) \end{array} \right\}$$

Experiment $\text{Exp}_{\text{corr}}^{\text{HSS}}$

$\text{Exp}_{\text{corr}}^{\text{HSS}}(1^\lambda, x_1, \dots, x_n) : //$ Correctness Experiment

1. $\text{sk} \stackrel{\$}{\leftarrow} \text{HSS.Setup}(1^\lambda)$
2. For $i \in [n]$, $(x_i^{(1)}, \dots, x_i^{(\ell)}) \stackrel{\$}{\leftarrow} \text{Share}(1^\lambda, \text{sk}, x_i)$
3. For $j \in [\ell]$, $z^{(j)} \leftarrow \text{Eval}(j, (x_1^{(j)}, \dots, x_n^{(j)}), C)$
4. Output $z \leftarrow \sum_{j \in [\ell]} z^{(j)}$

Figure 10: Correctness of the notion of secret-key HSS of Definition 21

Experiment $\text{Real}_{\mathcal{Z}}$ and $\text{Ideal}_{\mathcal{Z}}$

$\text{Real}_{\mathcal{Z}}(1^\lambda, x_1, \dots, x_q) : // \text{ Security: Real World}$

1. $\text{sk} \xleftarrow{\$} \text{HSS.Setup}(1^\lambda)$
2. For $k \in [q]$, $(x_k^{(1)}, \dots, x_k^{(\ell)}) \xleftarrow{\$} \text{Share}(1^\lambda, \text{sk}, x_k)$
3. Output $(x_k^{(j)})_{k \in [q], j \in \mathcal{Z}}$

$\text{Ideal}_{\mathcal{Z}}(1^\lambda, q) : // \text{ Security: Ideal World}$

Output $\text{Sim}(1^\lambda, \mathcal{Z}, q)$

Figure 11: Experiments for security in the notion of secret-key HSS of Definition 21

A.2 Semi-private “high-end” HSS.

Definition 22 (“High-end” semi-private HSS). Let $\ell \in \mathbb{N} \setminus \{0, 1\}$. Let $\mathcal{R} = \mathcal{R}(\lambda)$ be a ring and $n_1, \dots, n_\ell \in \text{poly}(\lambda)$. An ℓ -party additive semi-private HSS scheme supporting a class of computations $\mathcal{C} \subseteq \mathcal{R}^{n_1} \times \dots \times \mathcal{R}^{n_\ell} \rightarrow \mathcal{R}$ is a triple of polynomial-time algorithms $\text{HSS} = (\text{Setup}, \text{Share}^{\text{sp}}, \text{Eval})$, where Setup and Share^{sp} are probabilistic but Eval is deterministic, with the following syntax:

- $\text{HSS.Setup}(1^\lambda)$: On input the security parameter 1^λ , the setup algorithm Setup outputs a secret key sk .
- $\text{HSS.Share}^{\text{sp}}(1^\lambda, \text{sk}, j, y)$: On input the security parameter 1^λ , a secret key sk , a party index j , and a semi-private input $y \in \mathcal{R}$, the semi-private input sharing algorithm Share^{sp} outputs an ℓ -tuple of input shares $(y^{(1)}, \dots, y^{(\ell)})$.
- $\text{HSS.Eval}(j, ((x_{1,i}^{(j)})_{i \in [n_1]}, \dots, (x_{\ell,i}^{(j)})_{i \in [n_\ell]}), C)$: On input a party index j , a collection of input shares $((x_{1,i}^{(j)})_{i \in [n_1]}, \dots, (x_{\ell,i}^{(j)})_{i \in [n_\ell]})$, and a supported evaluation circuit $C \in \mathcal{C}$, the homomorphic evaluation algorithm Eval returns the j^{th} output share $y^{(j)} \in \mathcal{R}$.

We additionally require that HSS satisfy the following notions of correctness and *semi-private* security.

- **Correctness.** For any sufficiently large $\lambda \in \mathbb{N}$, for any circuit $C \in \mathcal{C}$, any C -admissible inputs $(x_1, \dots, x_\ell) \in \mathcal{R}^{n_1} \times \dots \times \mathcal{R}^{n_\ell}$,

$$\Pr \left[y = C(x_1, \dots, x_\ell) : y \xleftarrow{\$} \text{Exp}_{\text{corr}}^{\text{HSS}}(1^\lambda, x_1, \dots, x_\ell) \right] \geq 1 - \text{negl}(\lambda)$$

where $\text{Exp}_{\text{corr}}^{\text{HSS}}$ is defined in Fig. 12.

- **Semi-Private Security.** There exists a *p.p.t.* algorithm Sim^{sp} (a simulator) such that, for every set of corrupted parties $\mathcal{Z} \subseteq [\ell]$, every $q \in \text{poly}(\lambda)$, and any $(j_1, x_1), \dots, (j_q, x_q) \in [\ell] \times \mathcal{R}$, the outputs of the experiments $\text{Real}_{\mathcal{Z}}^{\text{sp}}(1^\lambda, (j_k, x_k)_{k \in [q]})$ and $\text{Ideal}_{\mathcal{Z}}^{\text{sp}}(1^\lambda, (j_k)_{k \in [q]}, (x_k)_{k: j_k \in \mathcal{Z}})$ (Fig. 13) are computationally indistinguishable.⁹

If HSS satisfies the following *optional* property, we say that HSS is *offline-online with respect to the first party's semi-private inputs*.

- **Offline-online sharing for P_1 -semi-private inputs.** We say the semi-private input sharing algorithm Share^{sp} is *offline-online* for P_1 -semi-private inputs if there exist *p.p.t.* algorithms $\text{Share}_1^{\text{sp}}, \dots, \text{Share}_\ell^{\text{sp}}$ with the following syntax and properties:
 - For $i \in [2, \ell]$, $\text{Share}_i^{\text{sp}}(1^\lambda, \text{sk}, \text{aux}_\ell, \dots, \text{aux}_{i+1})$: On input the security parameter 1^λ , the secret key sk , and the auxiliary information $(\text{aux}_j)_{j \in [i+1, \ell]}$ output by the input-sharing algorithms of the parties with a larger index (for $i = \ell$, there is none), the i^{th} input-sharing algorithm $\text{Share}_i^{\text{sp}}$ produces an i^{th} input share $x^{(i)}$ and auxiliary information aux_i . The offline algorithms are thus run in decreasing order of party index, $\text{Share}_\ell^{\text{sp}}$ first. This is the order in which auxiliary information needs to flow in our recursive construction (Fig. 15), where the last party's offline algorithm produces auxiliary information consumed by all the others.
 - $\text{Share}_1^{\text{sp}}(1^\lambda, \text{sk}, \text{aux}_2, \dots, \text{aux}_\ell, x)$: On input the security parameter 1^λ , the secret key sk , auxiliary information $(\text{aux}_i)_{i \in [2, \ell]}$, and a semi-private input x , the first input-sharing algorithm $\text{Share}_1^{\text{sp}}$ produces a first input share $x^{(1)}$.
 - **Offline-online correctness.** For any sufficiently large $\lambda \in \mathbb{N}$, for any sk in the support of $\text{Setup}(1^\lambda)$, and for any $x \in \mathcal{R}(\lambda)$, the following distributions are identically distributed:

$$\left\{ (x^{(1)}, \dots, x^{(\ell)}) : (x^{(1)}, \dots, x^{(\ell)}) \stackrel{\$}{\leftarrow} \text{Share}^{\text{sp}}(1^\lambda, \text{sk}, 1, x) \right\}$$

and

$$\left\{ (x^{(1)}, \dots, x^{(\ell)}) : \begin{array}{ll} (x^{(\ell)}, \text{aux}_\ell) & \stackrel{\$}{\leftarrow} \text{Share}_\ell^{\text{sp}}(1^\lambda, \text{sk}) \\ (x^{(\ell-1)}, \text{aux}_{\ell-1}) & \stackrel{\$}{\leftarrow} \text{Share}_{\ell-1}^{\text{sp}}(1^\lambda, \text{sk}, \text{aux}_\ell) \\ & \dots \\ (x^{(2)}, \text{aux}_2) & \stackrel{\$}{\leftarrow} \text{Share}_2^{\text{sp}}(1^\lambda, \text{sk}, \text{aux}_\ell, \dots, \text{aux}_3) \\ x^{(1)} & \stackrel{\$}{\leftarrow} \text{Share}_1^{\text{sp}}(1^\lambda, \text{sk}, \text{aux}_2, \dots, \text{aux}_\ell, x) \end{array} \right\}$$

⁹As in the correctness experiment, all inputs are shared under the same secret key. The simulator is given the inputs that are semi-private to a corrupted party, which the adversary may learn; the case $q = 1$ with $j_1 \notin \mathcal{Z}$ is the usual single-input notion.

Experiment $\text{Exp}_{\text{corr}}^{\text{HSS}}$

$\text{Exp}_{\text{corr}}^{\text{HSS}}(1^\lambda, x_1, \dots, x_\ell) : //$ Correctness Experiment

1. $\text{sk} \xleftarrow{\$} \text{HSS.Setup}(1^\lambda)$
2. For $j \in [\ell]$:
 - Parse x_j as $x_j = (x_{j,i})_{i \in [n_j]}$
 - For $i \in [n_j]$, $(x_{j,i}^{(1)}, \dots, x_{j,i}^{(\ell)}) \xleftarrow{\$} \text{Share}^{\text{SP}}(1^\lambda, \text{sk}, j, x_{j,i})$
3. For $j \in [\ell]$, $z^{(j)} \leftarrow \text{Eval}(j, ((x_{1,i}^{(j)})_{i \in [n_1]}, \dots, (x_{\ell,i}^{(j)})_{i \in [n_\ell]}), C)$
4. Output $z \leftarrow \sum_{j \in [\ell]} z^{(j)}$

Figure 12: Correctness of the notion of secret-key HSS of Definition 22

Experiment $\text{Real}_{\mathcal{Z}}^{\text{SP}}$ and $\text{Ideal}_{\mathcal{Z}}^{\text{SP}}$

$\text{Real}_{\mathcal{Z}}^{\text{SP}}(1^\lambda, (j_k, x_k)_{k \in [q]}) : //$ Semi-private security: Real World

1. $\text{sk} \xleftarrow{\$} \text{HSS.Setup}(1^\lambda)$
2. For $k \in [q]$, $(x_k^{(1)}, \dots, x_k^{(\ell)}) \xleftarrow{\$} \text{Share}^{\text{SP}}(1^\lambda, \text{sk}, j_k, x_k)$
3. Output $(x_k^{(i)})_{k \in [q], i \in \mathcal{Z}}$

$\text{Ideal}_{\mathcal{Z}}^{\text{SP}}(1^\lambda, (j_k)_{k \in [q]}, (x_k)_{k: j_k \in \mathcal{Z}}) : //$ Semi-private security: Ideal World

Output $\text{Sim}^{\text{SP}}(1^\lambda, \mathcal{Z}, (j_k)_{k \in [q]}, (x_k)_{k: j_k \in \mathcal{Z}})$

Figure 13: Experiments for semi-private security in the notion of secret-key HSS of Definition 22

B Building multiparty “high-end” HSS

In Section 4 we provided a recursive construction of multiparty semi-private “low-end” HSS. In Appendix B.1, we note that this recursion also works for “high-end” HSS.

In Section 5 we provided a reduction from semi-private to private multiparty “low-end” HSS. In Appendix B.2, we note that this reduction also works for “high-end” HSS.

B.1 Semi-private “high-end” HSS: from two-party to multi-party

“High-end” d -HSS

// d -party semi-private “high-end” HSS (as per Definition 22) for the class $\mathcal{C}_d$ (Definition 8), as parameterised by input sizes $n_1, \dots, n_d$

Requires:

- The classes $\mathcal{C}_2$, $\mathcal{C}_{d-1}$, and $\mathcal{C}_d$ should all be parameterised by the same subset $S \subseteq \mathcal{R}$ of admissible inputs, or by none at all. If S is defined, we require $\{0_{\mathcal{R}}, 1_{\mathcal{R}}\} \subseteq S$.
- $\text{HSS}_{d-1} = (\text{Setup}_{d-1}, \text{Share}_{d-1}, \text{Eval}_{d-1})$ is a “high-end” $(d-1)$ -party semi-private HSS (as per Definition 22) for the class $\mathcal{C}_{d-1}$ (Definition 8), as parameterised by input sizes $n_1, \dots, n_{d-1}$.
- Let $M(\lambda)$ be a polynomial upper bound on the bit length of each share output by $\text{Share}_{d-1}(1^\lambda, \cdot)$ on admissible inputs, and $L(\lambda)$ a polynomial upper bound on the bit length of the outputs of Eval_{d-1} . Set $m := M$, let $\text{Encode}: \{0, 1\}^M \rightarrow \mathcal{R}^m$, $(b_1, \dots, b_M) \mapsto (b_1 \cdot 1_{\mathcal{R}}, \dots, b_M \cdot 1_{\mathcal{R}})$, and let $\text{Decode}: \{0_{\mathcal{R}}, 1_{\mathcal{R}}\}^M \rightarrow \{0, 1\}^M$ be its inverse; fix $\rho_1, \dots, \rho_L \in \mathcal{R}$ such that every output y of Eval_{d-1} has a binary representation $(b_k)_{k \in [L]}$ with $y = \sum_k b_k \rho_k$ (as in Fig. 6).
- For $i \in [d-1]$, $\text{HSS}_{2,i} = (\text{Setup}_{2,i}, \text{Share}_{2,i}, \text{Eval}_{2,i})$ is a “high-end” 2-party semi-private HSS (as per Definition 22) for the class $\mathcal{C}_2$ (Definition 8), as parameterised by input sizes m, n_d .
- *Convention.* Since Definition 22 shares one element of $\mathcal{R}$ per call, applying Share_{d-1} or $\text{Share}_{2,i}$ to a vector denotes independent calls on its coordinates, the shares being collected coordinate-wise; likewise, running an offline sharing algorithm of HSS_{d-1} or $\text{HSS}_{2,i}$ for an input of s coordinates denotes s independent runs, whose shares and auxiliary information are collected as s -tuples. Party i ’s share of a vector is thus the tuple of its shares of the coordinates, which is the collection that Eval of Definition 22 expects.
- For $i \in [d-1]$, $g_i: \mathcal{R}^m \rightarrow \mathcal{R}$ is the $\mathcal{R}$ -arithmetic circuit obtained, exactly as in Fig. 6, by arithmetically emulating a boolean circuit $C_i: \{0, 1\}^M \rightarrow \{0, 1\}^L$ computing $X \mapsto \text{Eval}_{d-1}(i, \text{Decode}(X), (f_1, \dots, f_{d-1}))$ and outputting $\sum_{k \in [L]} \rho_k \cdot C_i(X)_k$; on inputs in $\{0_{\mathcal{R}}, 1_{\mathcal{R}}\}^m$ it computes $\text{Eval}_{d-1}(i, \text{Decode}(\cdot), (f_1, \dots, f_{d-1}))$ exactly.

$\text{HSS}_d.\text{Setup}(1^\lambda) :$

1. $\text{sk} \xleftarrow{\$} \text{Setup}_{d-1}(1^\lambda)$
2. For $i \in [d-1]$, $\text{sk}_i \xleftarrow{\$} \text{Setup}_{2,i}(1^\lambda)$
3. Output $(\text{sk}, \text{sk}_1, \dots, \text{sk}_{d-1})$

$\text{HSS}_d.\text{Share}(1^\lambda, \tilde{\text{sk}}, j, x) :$

- If $j \in [d-1]$:
 1. Parse $\tilde{\text{sk}} = (\text{sk}, \text{sk}_1, \dots, \text{sk}_{d-1})$
 2. $(w_1, \dots, w_{d-1}) \xleftarrow{\$} \text{Share}_{d-1}(1^\lambda, \text{sk}, j, x)$
 3. For $i \in [d-1]$:
 - (a) $\tilde{w}_i \leftarrow \text{Encode}(w_i)$
 - (b) $(z_{i,1}, z_{i,2}) \xleftarrow{\$} \text{Share}_{2,i}(1^\lambda, \text{sk}_i, 1, \tilde{w}_i)$
 - (c) $\text{share}_i \leftarrow z_{i,1}$
 4. $\text{share}_d \leftarrow (z_{j,2})_{j \in [d-1]}$
 5. Output $(\text{share}_1, \dots, \text{share}_d)$
- If $j = d$:
 1. Parse $\tilde{\text{sk}} = (\text{sk}, \text{sk}_1, \dots, \text{sk}_{d-1})$
 2. For $i \in [d-1]$:
 - (a) $(z_{i,1}, z_{i,2}) \xleftarrow{\$} \text{Share}_{2,i}(1^\lambda, \text{sk}_i, 2, x)$
 - (b) $\text{share}_i \leftarrow z_{i,1}$
 3. $\text{share}_d \leftarrow (z_{j,2})_{j \in [d-1]}$
 4. Output $(\text{share}_1, \dots, \text{share}_d)$

$\text{HSS}_d.\text{Eval}(i, \text{shares}_i, (f_1, \dots, f_d))$ // Unchanged from Fig. 6.

Figure 14: “High-end” semi-private d -HSS for $\mathcal{C}_d$ from “high-end” semi-private $(d-1)$ -HSS for $\mathcal{C}_{d-1}$ and “high-end” semi-private 2-HSS for $\mathcal{C}_2$.

Offline-Online Sharing

// Offline-Online sharing algorithms $\text{Share}_{d,1}, \text{Share}_{d,2}, \dots, \text{Share}_{d,d}$ for the construction of Fig. 14. As prescribed by Definition 22, they are run in the order $\text{Share}_{d,d}, \text{Share}_{d,d-1}, \dots, \text{Share}_{d,2}$, then $\text{Share}_{d,1}$.

Requires:

- This construction inherits the requirements of Fig. 14 (in particular its convention for vector inputs).

- $\text{Share}_{d-1,1}, \dots, \text{Share}_{d-1,d-1}$ are offline-online sharing algorithms for HSS_{d-1} , with respect to the first party's semi-private inputs.
- For $i \in [d-1]$, $\text{Share}_{2,i,1}$ and $\text{Share}_{2,i,2}$ are offline-online sharing algorithms for $\text{HSS}_{2,i}$, with respect to the first party's semi-private inputs. // Party 1 of $\text{HSS}_{2,i}$ is party i of HSS_d , and party 2 of $\text{HSS}_{2,i}$ is party d of HSS_d .

$\text{Share}_{d,d}(1^\lambda, \tilde{\text{sk}})$: // Party d is not a party of HSS_{d-1} . It runs first, and its auxiliary information is consumed by all the other parties.

1. Parse $\tilde{\text{sk}} = (\text{sk}, \text{sk}_1, \dots, \text{sk}_{d-1})$
2. For $i \in [d-1]$, $(z_{i,2}, \text{aux}_{2,i}) \xleftarrow{\$} \text{Share}_{2,i,2}(1^\lambda, \text{sk}_i)$ // m runs each, one per coordinate of $\tilde{w}_i$.
3. $\text{share}_d \leftarrow (z_{i,2})_{i \in [d-1]}$ and $\text{aux}_d \leftarrow (\text{aux}_{2,i})_{i \in [d-1]}$
4. Output $(\text{share}_d, \text{aux}_d)$

$\text{Share}_{d,i}(1^\lambda, \tilde{\text{sk}}, \text{aux}_d, \dots, \text{aux}_{i+1})$: (for $i \in [2, d-1]$)

1. Parse $\tilde{\text{sk}} = (\text{sk}, \text{sk}_1, \dots, \text{sk}_{d-1})$ and aux_d as $(\text{aux}_{2,j})_{j \in [d-1]}$ // $\text{aux}_{i+1}, \dots, \text{aux}_{d-1}$ are the auxiliary information output by parties $i+1, \dots, d-1$ of HSS_{d-1} .
2. $(w_i, \text{aux}_{d-1,i}) \xleftarrow{\$} \text{Share}_{d-1,i}(1^\lambda, \text{sk}, \text{aux}_{d-1}, \dots, \text{aux}_{i+1})$ // n_1 runs, one per coordinate of party 1's input; for $i = d-1$ there is no auxiliary input.
3. $\tilde{w}_i \leftarrow \text{Encode}(w_i)$
4. $z_{i,1} \xleftarrow{\$} \text{Share}_{2,i,1}(1^\lambda, \text{sk}_i, \text{aux}_{2,i}, \tilde{w}_i)$
5. $\text{share}_i \leftarrow z_{i,1}$ and $\text{aux}_i \leftarrow \text{aux}_{d-1,i}$
6. Output $(\text{share}_i, \text{aux}_i)$

$\text{Share}_{d,1}(1^\lambda, \tilde{\text{sk}}, \text{aux}_2, \dots, \text{aux}_d, x)$:

1. Parse $\tilde{\text{sk}} = (\text{sk}, \text{sk}_1, \dots, \text{sk}_{d-1})$ and aux_d as $(\text{aux}_{2,j})_{j \in [d-1]}$
2. $w_1 \xleftarrow{\$} \text{Share}_{d-1,1}(1^\lambda, \text{sk}, \text{aux}_2, \dots, \text{aux}_{d-1}, x)$
3. $\tilde{w}_1 \leftarrow \text{Encode}(w_1)$
4. $z_{1,1} \xleftarrow{\$} \text{Share}_{2,1,1}(1^\lambda, \text{sk}_1, \text{aux}_{2,1}, \tilde{w}_1)$
5. Output $\text{share}_1 \leftarrow z_{1,1}$

Figure 15: Offline-Online sharing algorithms $\text{Share}_{d,1}, \text{Share}_{d,2}, \dots, \text{Share}_{d,d}$ for the construction of Fig. 14.

Theorem 23. *If its requirements are met, then the construction of Fig. 14 is offline-online*

for the first party's semi-private inputs. Furthermore, the algorithms of Fig. 15 provide an explicit offline-online sharing mechanism for it.

Proof. Fix $\tilde{\mathbf{sk}} = (\mathbf{sk}, \mathbf{sk}_1, \dots, \mathbf{sk}_{d-1})$ in the support of $\text{HSS}_d.\text{Setup}(1^\lambda)$ and $x \in \mathcal{R}^{n_1}$. Recall that $\text{HSS}_d.\text{Share}(1^\lambda, \tilde{\mathbf{sk}}, 1, x)$ runs $\text{Share}_{d-1}(1^\lambda, \mathbf{sk}, 1, x)$ and then, independently for each $i \in [d-1]$, $\text{Share}_{2,i}(1^\lambda, \mathbf{sk}_i, 1, \tilde{w}_i)$ (coordinate-wise in both cases, by the convention of Fig. 14). Consider the composition $\text{Share}_{d,d}, \text{Share}_{d,d-1}, \dots, \text{Share}_{d,2}, \text{Share}_{d,1}$ of Fig. 15. First, $(w_1, \dots, w_{d-1})$ is produced, coordinate by coordinate, by the chain $\text{Share}_{d-1,d-1}, \dots, \text{Share}_{d-1,2}, \text{Share}_{d-1,1}$ in the order prescribed by Definition 22, so it is distributed identically to the output of $\text{Share}_{d-1}(1^\lambda, \mathbf{sk}, 1, x)$ by offline-online correctness of HSS_{d-1} . Second, conditioned on $(w_i)_{i \in [d-1]}$, each pair $(z_{i,1}, z_{i,2})$ is produced by $\text{Share}_{2,i,2}(1^\lambda, \mathbf{sk}_i)$ followed by $\text{Share}_{2,i,1}(1^\lambda, \mathbf{sk}_i, \text{aux}_{2,i}, \tilde{w}_i)$, which is distributed identically to $\text{Share}_{2,i}(1^\lambda, \mathbf{sk}_i, 1, \tilde{w}_i)$ by offline-online correctness of $\text{HSS}_{2,i}$; conditioning is legitimate since $\text{Share}_{2,i,2}$ is run before, and independently of, $\tilde{w}_i$, and the $d-1$ pairs are independent given $(w_i)_{i \in [d-1]}$ in both distributions. The two distributions of Definition 22 therefore coincide. Note that every auxiliary value consumed by party i is produced by a party of larger index ($\text{aux}_{2,i}$ by party d , and $\text{aux}_{d-1,j}$ by party $j > i$), which is why Definition 22 fixes the decreasing order, and that only the P_1 property of HSS_{d-1} and of the $\text{HSS}_{2,i}$ is used: the branch $j = d$ of $\text{HSS}_d.\text{Share}$ is never run offline. $\square$

B.2 From semi-private to private “high-end” HSS

“High-end” HSS

// n -party “high-end” HSS (as per Definition 21, instantiated with $\ell \leftarrow n$ parties and $n \leftarrow m$ inputs) for the class of degree- d m -variate functions with polynomially bounded Waring rank.

Requires:

- $\text{spHSS} = (\text{Setup}^{\text{sp}}, \text{Share}^{\text{sp}}, \text{Eval}^{\text{sp}})$ is an n -party “high-end” semi-private HSS scheme (as per Definition 22) for the class $\mathcal{C}_n$ (Definition 8), parameterised as in Fig. 8: $\ell \leftarrow n$, $\mathcal{R} \leftarrow \mathcal{R}^{(d+1) \times (d+1)}$, $n_1, \dots, n_\ell \leftarrow m$, and admissible inputs $S \leftarrow \text{scalar-to-matrix}(\mathcal{R})$.
- **scalar-to-matrix**: $\mathcal{R} \rightarrow \mathcal{R}^{(d+1) \times (d+1)}$ maps a scalar x to the $(d+1)$ -by- $(d+1)$ matrix with x in position $(0, 0)$, and 0 elsewhere.
- **PRF** is a pseudorandom function with key space $\{0, 1\}^\lambda$, input space $\{0, 1\}^\lambda$, and outputs in $\mathcal{R}^{n-1} \times (\{0, 1\}^\lambda)^{n-1}$

$\text{HSS}.\text{Setup}(1^\lambda)$:

1. $\mathbf{sk}^{\text{sp}} \xleftarrow{\$} \text{Setup}^{\text{sp}}(1^\lambda)$
2. $k \xleftarrow{\$} \{0, 1\}^\lambda$ // This is a PRF key
3. Output $(\mathbf{sk}^{\text{sp}}, k)$

$\text{HSS}.\text{Share}(1^\lambda, \mathbf{sk}, x)$:

1. Parse $\mathbf{sk}$ as $(\mathbf{sk}^{\text{sp}}, k)$

2. Sample $\text{id} \xleftarrow{\$} \{0, 1\}^\lambda$ // Serves as a unique session id^a used to re-derive the shares of x
3. $(x_2, \dots, x_n, R_2, \dots, R_n) \leftarrow \text{PRF}(k, \text{id})$
4. $x_1 \leftarrow x - \sum_{i=2}^n x_i$
5. $X_1 \leftarrow \text{scalar-to-matrix}(x_1)$ and $(\text{share}_1^{(1)}, \dots, \text{share}_1^{(n)}) \xleftarrow{\$} \text{spHSS.Share}^{\text{sp}}(1^\lambda, \text{sk}^{\text{sp}}, 1, X_1)$
// Fresh randomness. The sharing of X_1 is the only one that is not re-derivable from (k, id) ; this is what allows it to be produced online in Fig. 17, and it is needed for the two distributions of offline-online correctness to be identical.
6. For $i \in [2, n]$:
 - (a) $X_i \leftarrow \text{scalar-to-matrix}(x_i)$
 - (b) $(\text{share}_i^{(1)}, \dots, \text{share}_i^{(n)}) \leftarrow \text{spHSS.Share}^{\text{sp}}(1^\lambda, \text{sk}^{\text{sp}}, i, X_i; R_i)$
// Use random tape R_i . Without essentially any loss of generality, we can assume $\text{spHSS.Share}^{\text{sp}}$ indeed uses only λ bits of randomness.
7. For $i \in [n]$:

Set $\text{share}_i \leftarrow (\text{share}_j^{(i)})_{j \in [n]}$
8. Output $(\text{share}_1, \dots, \text{share}_n)$

$\text{HSS.Eval}(i, \text{share}_i, f)$ // Unchanged from Fig. 8, up to regrouping the shares: for each of the m inputs x_t ($t \in [m]$), party i holds the tuple $(\text{share}_{t,p}^{(i)})_{p \in [n]}$ output by Share ; for each $p \in [n]$, the collection $(\text{share}_{t,p}^{(i)})_{t \in [m]}$ is party i 's spHSS share of the m -vector of matrices semi-private to party p , which is the form expected by $\text{spHSS.Eval}^{\text{sp}}$.

^aWe sample the identifiers at random rather than keep a stateful counter; the only property we use is that they are pairwise distinct, which holds with all but negligible probability.

Figure 16: “High-end” n -HSS for functions of polynomially bounded degree and mixed Waring rank, from “high-end” n - spHSS for $\mathcal{C}_n$.

Offline-Online Sharing

// Offline-Online sharing algorithms $\text{Share}_1, \text{Share}_2, \dots, \text{Share}_n$ for the construction of Fig. 16. As prescribed by Definition 21, they are run in the order $\text{Share}_n, \text{Share}_{n-1}, \dots, \text{Share}_2$, then Share_1 .

Requires:

- This construction inherits the requirements of Fig. 16.
- $\text{Share}_1^{\text{sp}}, \dots, \text{Share}_n^{\text{sp}}$ are offline-online sharing algorithms for spHSS (with respect to the first party's semi-private inputs).

$\text{HSS.Share}_n(1^\lambda, \text{sk})$: // The “session id” id is sampled by Share_n , which runs first, and passed down to $\text{Share}_{n-1}, \dots, \text{Share}_2$ and then to Share_1 via aux_n , which is why we define Share_n separately.

1. Parse sk as $(\text{sk}^{\text{sp}}, k)$
2. Sample $\text{id} \xleftarrow{\$} \{0, 1\}^\lambda$ // Same role as in HSS.Share ; it is the only fresh randomness shared with the other sharing algorithms.
3. $(x_2, \dots, x_n, R_2, \dots, R_n) \leftarrow \text{PRF}(k, \text{id})$
4. For $j \in [2, n]$:
 - (a) $X_j \leftarrow \text{scalar-to-matrix}(x_j)$
 - (b) $(\text{share}_j^{(1)}, \dots, \text{share}_j^{(n)}) \leftarrow \text{spHSS.Share}^{\text{sp}}(1^\lambda, \text{sk}^{\text{sp}}, j, X_j; R_j)$
5. $(\text{share}_1^{(n)}, \text{aux}_n^{\text{sp}}) \xleftarrow{\$} \text{Share}_n^{\text{sp}}(1^\lambda, \text{sk}^{\text{sp}})$
6. Set $\text{share}_n \leftarrow (\text{share}_j^{(n)})_{j \in [n]}$ and $\text{aux}_n \leftarrow (\text{id}, \text{aux}_n^{\text{sp}})$
7. Output $(\text{share}_n, \text{aux}_n)$

$\text{HSS.Share}_i(1^\lambda, \text{sk}, \text{aux}_n, \dots, \text{aux}_{i+1})$: (for $i \in [2, n-1]$)

1. Parse sk as $(\text{sk}^{\text{sp}}, k)$ and aux_n as $(\text{id}, \text{aux}_n^{\text{sp}})$ // id is read from aux_n ; the auxiliary information $\text{aux}_n^{\text{sp}}, \text{aux}_{n-1}, \dots, \text{aux}_{i+1}$ is forwarded to $\text{Share}_i^{\text{sp}}$.
2. $(x_2, \dots, x_n, R_2, \dots, R_n) \leftarrow \text{PRF}(k, \text{id})$
3. For $j \in [2, n]$:
 - (a) $X_j \leftarrow \text{scalar-to-matrix}(x_j)$
 - (b) $(\text{share}_j^{(1)}, \dots, \text{share}_j^{(n)}) \leftarrow \text{spHSS.Share}^{\text{sp}}(1^\lambda, \text{sk}^{\text{sp}}, j, X_j; R_j)$
4. $(\text{share}_1^{(i)}, \text{aux}_i) \xleftarrow{\$} \text{Share}_i^{\text{sp}}(1^\lambda, \text{sk}^{\text{sp}}, \text{aux}_n^{\text{sp}}, \text{aux}_{n-1}, \dots, \text{aux}_{i+1})$
5. Set $\text{share}_i \leftarrow (\text{share}_j^{(i)})_{j \in [n]}$
6. Output $(\text{share}_i, \text{aux}_i)$

$\text{HSS.Share}_1(1^\lambda, \text{sk}, \text{aux}_2, \dots, \text{aux}_n, x)$:

1. Parse sk as $(\text{sk}^{\text{sp}}, k)$ and aux_n as $(\text{id}, \text{aux}_n^{\text{sp}})$
2. $(x_2, \dots, x_n, R_2, \dots, R_n) \leftarrow \text{PRF}(k, \text{id})$
3. For $j \in [2, n]$:
 - (a) $X_j \leftarrow \text{scalar-to-matrix}(x_j)$
 - (b) $(\text{share}_j^{(1)}, \dots, \text{share}_j^{(n)}) \leftarrow \text{spHSS.Share}^{\text{sp}}(1^\lambda, \text{sk}^{\text{sp}}, j, X_j; R_j)$
4. $x_1 \leftarrow x - \sum_{i=2}^n x_i$ and $X_1 \leftarrow \text{scalar-to-matrix}(x_1)$
5. $\text{share}_1^{(1)} \xleftarrow{\$} \text{Share}_1^{\text{sp}}(1^\lambda, \text{sk}^{\text{sp}}, \text{aux}_2, \dots, \text{aux}_{n-1}, \text{aux}_n^{\text{sp}}, X_1)$
6. Set $\text{share}_1 \leftarrow (\text{share}_j^{(1)})_{j \in [n]}$
7. Output share_1

Figure 17: Offline-Online sharing algorithms for the construction of Fig. 16.

Theorem 24. *Let $\mathcal{R}(\lambda)$ be a finite commutative ring. If its requirements are met and PRF is a pseudorandom function, then the construction of Fig. 16 is an n -party “high-end” HSS (as per Definition 21, instantiated with $\ell \leftarrow n$ parties and $n \leftarrow m$ inputs) for the class of all functions $\mathcal{R}^m \rightarrow \mathcal{R}$ with degree d and mixed-degree Waring rank (Definition 6) at most r . If moreover spHSS is offline-online for P_1 -semi-private inputs, then HSS is offline-online, with the algorithms of Fig. 17.*

Proof. Sharing an input costs one evaluation of PRF and n calls to $\text{spHSS.Share}^{\text{sp}}$, homomorphic evaluation costs r calls to $\text{spHSS.Eval}^{\text{sp}}$, and party i ’s share of an input consists of n shares of spHSS. Throughout, we follow the convention of Fig. 16 that each of the m inputs is shared by a separate call to HSS.Share ; we write $x_{t,p}$ for the p^{th} additive share of the t^{th} input x_t (so that $x_t = \sum_{p \in [n]} x_{t,p}$), $X_{t,p} \leftarrow \text{scalar-to-matrix}(x_{t,p})$, and $\text{share}_{t,p}^{(i)}$ for party i ’s spHSS share of $X_{t,p}$, which is a semi-private input of party p .

- **Correctness.** Let $f: \mathcal{R}^m \rightarrow \mathcal{R}$ be in the class and let $(x_1, \dots, x_m) \in \mathcal{R}^m$. Consider the probability experiment of $\text{Exp}_{\text{corr}}^{\text{HSS}}(1^\lambda, x_1, \dots, x_m)$ (Fig. 10). Whatever the outputs of PRF, each call to HSS.Share sets $x_{t,1} \leftarrow x_t - \sum_{p=2}^n x_{t,p}$, so $\sum_{p \in [n]} x_{t,p} = x_t$, and every $X_{t,p}$ lies in the admissible set $\text{scalar-to-matrix}(\mathcal{R})$. After regrouping (c.f. the description of HSS.Eval in Fig. 16), party i holds, for each $p \in [n]$, its spHSS share of the collection $(X_{t,p})_{t \in [m]}$ of m matrices semi-private to party p ; this is exactly the input on which HSS.Eval of Fig. 8 is run, with $x^{(p)} \leftarrow (x_{t,p})_{t \in [m]}$ and $\sum_{p \in [n]} x^{(p)} = x$. By correctness of spHSS (Definition 22, whose correctness experiment shares the coordinates of each party’s input one by one, as HSS.Share does), the rest of the argument is identical to the correctness proof of Theorem 16.
- **Security.** Let Sim^{sp} be a simulator (as per Fig. 13) for spHSS. Let $\mathcal{Z} \subsetneq [n]$ be the set of corrupted parties (recall that Definition 21 only quantifies over $\mathcal{Z} \subsetneq [n]$, i.e. there is at least one honest party), let $q \in \text{poly}(\lambda)$, and let $x_1, \dots, x_q \in \mathcal{R}$ be the inputs shared in the security experiment. Consider the following sequence of hybrids:
 1. **Real world:** This is the experiment $\text{Real}_{\mathcal{Z}}$ of Fig. 11. In particular, note that it contains, for each $t \in [q]$, the instructions “Sample $\text{id}_t \xleftarrow{\$} \{0,1\}^\lambda$ ” and “ $(x_{t,2}, \dots, x_{t,n}, R_{t,2}, \dots, R_{t,n}) \leftarrow \text{PRF}(k, \text{id}_t)$ ”, all under the same PRF key k .
 2. **Hybrid $\mathcal{H}_1$:** In this hybrid, every evaluation of $\text{PRF}(k, \cdot)$ is replaced with an evaluation of a uniformly random function $F: \{0,1\}^\lambda \rightarrow \mathcal{R}^{n-1} \times (\{0,1\}^\lambda)^{n-1}$. This hybrid is indistinguishable from the previous one by pseudorandomness of PRF (the reduction makes q oracle queries).
 3. **Hybrid $\mathcal{H}_2$:** In this hybrid, the experiment aborts if $\text{id}_t = \text{id}_{t'}$ for some $t \neq t'$. This hybrid is statistically indistinguishable from the previous one, since the identifiers are uniform and the abort probability is at most $q^2/2^\lambda$. Conditioned on not aborting, the values $(x_{t,2}, \dots, x_{t,n}, R_{t,2}, \dots, R_{t,n})_{t \in [q]}$ are independent and uniform, so the $x_{t,p}$ ($p \in [2, n]$) are fresh uniform additive shares and each “ $(\text{share}_{t,p}^{(1)}, \dots, \text{share}_{t,p}^{(n)}) \leftarrow \text{spHSS.Share}^{\text{sp}}(1^\lambda, \text{sk}^{\text{sp}}, p, X_{t,p}; R_{t,p})$ ” is a fresh, honestly randomised sharing (for $p = 1$ this already holds in the real world).

4. **Hybrid $\mathcal{H}_3$:** In this hybrid, the instructions computing $(x_{t,p})_{p \in \mathcal{Z}}$ from x_t are replaced with " $(x_{t,p})_{p \in \mathcal{Z}} \xleftarrow{\$} \mathcal{R}^{\mathcal{Z}}$ ", and the values x_t are no longer used. This hybrid is perfectly indistinguishable from the previous one: since $\mathcal{Z} \subsetneq [n]$, some party p^* is honest, and $(x_{t,p})_{p \in \mathcal{Z}}$ are $|\mathcal{Z}|$ of the n uniform additive shares of x_t over $\mathcal{R}$, hence uniform and independent of x_t (if $1 \in \mathcal{Z}$, this uses $p^* \geq 2$).
5. **Hybrid $\mathcal{H}_4$:** In this hybrid, the qn instructions " $(\text{share}_{t,p}^{(1)}, \dots, \text{share}_{t,p}^{(n)}) \leftarrow \text{spHSS.Share}^{\text{sp}}(1^\lambda, \text{sk}^{\text{sp}}, p, X_{t,p}; R_{t,p})$ " are replaced with the single instruction " $(\text{share}_{t,p}^{(i)})_{t \in [q], p \in [n], i \in \mathcal{Z}} \xleftarrow{\$} \text{Sim}^{\text{sp}}(1^\lambda, \mathcal{Z}, (p)_{t \in [q], p \in [n]}, (X_{t,p})_{t \in [q], p \in \mathcal{Z}})$ ", and the honest parties' shares are dropped (they are not part of the output). This hybrid is indistinguishable from the previous one by semi-private security of spHSS (Definition 22), invoked *once*, with respect to corrupted parties $\mathcal{Z}$, on the qn queries $(p, X_{t,p})_{t \in [q], p \in [n]}$: all these sharings are computed under the same key sk^{sp} , which is why we need the multi-input game with arbitrary party indices rather than a hybrid over t (which would have to reprogram sk^{sp}). Note that the simulator is given precisely the inputs $X_{t,p}$ which are semi-private to a corrupted party, and that the previous hybrid samples these without reference to x_t .
6. **Ideal world:** This is the experiment $\text{Ideal}_{\mathcal{Z}}$ of Fig. 11, run with the simulator Sim of Fig. 18. It is code-equivalent to hybrid $\mathcal{H}_4$.

Simulator Sim

Requires:

- Sim inherits the requirements of HSS
- Sim^{sp} is a simulator for spHSS with respect to corrupted parties $\mathcal{Z}$.

Simulation: On input $(1^\lambda, \mathcal{Z}, q)$:

1. For $t \in [q]$ and $p \in \mathcal{Z}$:
 - (a) $x_{t,p} \xleftarrow{\$} \mathcal{R}$
 - (b) $X_{t,p} \leftarrow \text{scalar-to-matrix}(x_{t,p})$
2. $(\text{share}_{t,p}^{(i)})_{t \in [q], p \in [n], i \in \mathcal{Z}} \xleftarrow{\$} \text{Sim}^{\text{sp}}(1^\lambda, \mathcal{Z}, (p)_{t \in [q], p \in [n]}, (X_{t,p})_{t \in [q], p \in \mathcal{Z}})$
3. For $i \in \mathcal{Z}$ and $t \in [q]$, $\text{share}_t^{(i)} \leftarrow (\text{share}_{t,p}^{(i)})_{p \in [n]}$ // Party i 's share of the t^{th} input.
4. Output $(\text{share}_t^{(i)})_{t \in [q], i \in \mathcal{Z}}$

Figure 18: Simulator Sim for the construction of Fig. 16.

- **Offline-online correctness.** Assume spHSS is offline-online for P_1 -semi-private inputs, and fix $\text{sk} = (\text{sk}^{\text{sp}}, k)$ in the support of $\text{HSS.Setup}(1^\lambda)$ and $x \in \mathcal{R}$. In $\text{HSS.Share}(1^\lambda, \text{sk}, x)$ as well as in the composition $\text{Share}_n, \text{Share}_{n-1}, \dots, \text{Share}_2, \text{Share}_1$ of Fig. 17, the identifier id is uniform in $\{0, 1\}^\lambda$ and $(x_2, \dots, x_n, R_2, \dots, R_n)$ is the same deterministic function of

(k, id) ; the shares of $X_2, \dots, X_n$ (computed with the random tapes $R_2, \dots, R_n$) are therefore identically distributed in both. Conditioned on id , the shares of X_1 are output by $\text{spHSS.Share}^{\text{sp}}(1^\lambda, \text{sk}^{\text{sp}}, 1, X_1)$ with fresh randomness in HSS.Share , and by the composition $\text{Share}_n^{\text{sp}}, \dots, \text{Share}_2^{\text{sp}}, \text{Share}_1^{\text{sp}}$ (with the auxiliary information forwarded in the order prescribed by Definition 22) in Fig. 17; these are identically distributed by offline-online correctness of spHSS , applied to sk^{sp} and X_1 . This is where it matters that HSS.Share uses fresh randomness for X_1 rather than an output of PRF. Since, given id , the shares of X_1 are independent of those of $X_2, \dots, X_n$ in both distributions, the two distributions of Definition 21 coincide. $\square$

C Deferred proofs

C.1 Proof of Corollary 11

Proof of Corollary 11. HSS_N is obtained by recursively applying the construction of Fig. 6 $N - 2$ times, starting from HSS_2 . In particular, a straightforward hybrid argument handles correctness and security. It is also straightforward to check that, should the stated share sizes and computational complexity of Eval be correct, then $\text{HSS}_N.\text{Share}$ and $\text{HSS}_N.\text{Eval}$ run in polynomial time in the theorem's parameter ranges. This leaves us with proving the claimed efficiency bounds.

By Theorem 9 there is an absolute constant $c \geq 1$ such that for every $k \in [3, N]$,

$$\begin{aligned} S_k(s_1, \dots, s_k) &\leq ck \cdot S_2(S_{k-1}(s_1, \dots, s_{k-1}), s_k) \\ \text{CC}_k(s_1, \dots, s_k) &\leq ck \cdot \text{CC}_2(\text{CC}_{k-1}(s_1, \dots, s_{k-1}), s_k), \end{aligned}$$

and by Definition 10 there is a polynomial p such that $S_2(s_1, s_2) \leq \alpha s_1 + p(\lambda)s_2$ and $\text{CC}_2(s_1, s_2) \leq \beta s_1 + p(\lambda)s_2$.

- *Bounding S_N .* Let us now show by finite induction on $k = 2, \dots, N$ that $S_k(s_1, \dots, s_k) \leq A_k \cdot \sum_{i \in [k]} s_i$, where

$$A_k := \begin{cases} ck\alpha \cdot A_{k-1} & \text{if } k \geq 3 \\ \alpha \cdot p(\lambda) & \text{if } k = 2 \end{cases}$$

In particular, note that $A_N = \alpha p(\lambda) \prod_{k=3}^N (ck\alpha) \leq p(\lambda) c^N \alpha^N N^N$.

The induction's initialisation (*i.e.* $k = 2$) holds by inspection: $S_2(s_1, s_2) \leq \max(\alpha, p(\lambda)) \cdot (s_1 + s_2) \leq \alpha p(\lambda) \cdot (s_1 + s_2)$. For the inductive step, let $k \geq 3$ and assume the inequality holds up to $k - 1$. Observe that:

$$\begin{aligned} S_k(s_1, \dots, s_k) &\leq ck(\alpha \cdot S_{k-1}(s_1, \dots, s_{k-1}) + p(\lambda)s_k) \\ &\leq ck \left(\alpha A_{k-1} \sum_{i \in [k-1]} s_i + p(\lambda)s_k \right) \\ &\leq ck\alpha A_{k-1} \sum_{i \in [k]} s_i, \end{aligned}$$

where the last inequality uses $\alpha A_{k-1} \geq A_2 \geq p(\lambda)$.

Combining this inequality with $A_N \leq p(\lambda) c^N \alpha^N N^N$ yields the desired bound:

$$S_N(s_1, \dots, s_N) \leq \text{poly}(\lambda) 2^{O(N)} (\alpha N)^N \sum_{i \in [N]} s_i.$$

- *Bounding CC_N .* The desired bound on CC_N can be obtained completely analogously, except we replace α with β , and s_i with $|f_i|$.

□

C.2 Proof of Lemma 17

Proof of Lemma 17. First observe that, by the binomial theorem, $\forall a, b \in \mathcal{R}, M_a \cdot M_b = M_{a+b}$. Indeed,

$$\begin{aligned} (M_a \cdot M_b)[i][j] &= \sum_{k=0}^d M_a[i][k] \cdot M_b[k][j] \\ &= \sum_{k=0}^d \binom{k}{i} a^{k-i} \binom{j}{k} b^{j-k} \\ &= \sum_{k=i}^j \binom{k}{i} a^{k-i} \binom{j}{k} b^{j-k} \\ &= \sum_{k=0}^{j-i} \binom{k+i}{i} a^k \binom{j}{k+i} b^{j-k-i} \\ &= \sum_{k=0}^{j-i} \binom{j}{i} \binom{j-i}{k} a^k b^{j-k-i} // \binom{a}{b} \binom{b}{c} = \binom{a}{c} \binom{a-c}{b-c} \\ &= \binom{j}{i} (a+b)^{j-i} // \text{by the binomial theorem} \end{aligned}$$

So

$$\begin{aligned} &\underbrace{(1, 0, \dots, 0)}_{d+1} \cdot M_{a_1} \cdots M_{a_n} \cdot \underbrace{(0, \dots, 0, 1, 0, \dots, 0)}_{d+1}^{\overbrace{d'} \quad \overbrace{d-d'}}^\top \\ &= \underbrace{(1, 0, \dots, 0)}_{d+1} \cdot M_{a_1 + \dots + a_n} \cdot \underbrace{(0, \dots, 0, 1, 0, \dots, 0)}_{d+1}^{\overbrace{d'} \quad \overbrace{d-d'}}^\top \\ &= M_{a_1 + \dots + a_n}[0][d'] \\ &= (a_1 + \dots + a_n)^{d'} \end{aligned}$$

□

C.3 Proof of Theorem 19

Proof of Theorem 19. • **Correctness.** Fix $j \in [r]$ with $d_j \geq 1$. By the multinomial theorem, for all $y_1, \dots, y_n \in \mathcal{R}$,

$$(y_1 + \dots + y_n)^{d_j} = \sum_{\substack{k_1 + k_2 + \dots + k_n = d_j \\ k_1, \dots, k_n \geq 0}} \binom{d_j}{k_1, \dots, k_n} \prod_{i=1}^n y_i^{k_i}$$

Each monomial is characterised by a non-negative vector $(k_1, \dots, k_n)$ of L^1 -norm d_j . We can enumerate such vectors by first choosing the set S of non-zero positions (of size

between 1 and d_j), and then assigning each of those positions a positive weight such that all weights sum to d_j . Separating the sets of size one from the others:

$$\left(\sum_{i=1}^n y_i\right)^{d_j} = \sum_{i=1}^n y_i^{d_j} + \sum_{\substack{S \subseteq [n] \\ 2 \leq |S| \leq d_j}} \sum_{\substack{\vec{k} \in [1, d_j]^{|S|} \\ \|\vec{k}\|_1 = d_j}} \binom{d_j}{k_1, \dots, k_{|S|}} \prod_{\ell=1}^{|S|} y_{s_\ell}^{k_\ell} \quad (2)$$

where $s_1 < \dots < s_{|S|}$ denote the elements of S . Now, let $x \in \mathcal{R}^m$ and consider the experiment $\text{Exp}_{\text{corr}}^{\text{HSS}}(1^\lambda, x)$ of Fig. 2. Consider the random variables $(x^{(i)})_{i \in [n]}$, $(\text{share}_{i,S})_{S, i \in S}$, $(w_{i,j})_{i \in [n], j \in [r]}$, and $(y_i)_{i \in [n]}$ defined by inlining **Share** and **Eval** in the experiment, and, for $j \in [r]$, $S \subseteq [n]$ with $2 \leq |S| \leq d_j$, and $\vec{k} \in [1, d_j]^{|S|}$ with $\|\vec{k}\|_1 = d_j$, the event

$$E_{S,j,\vec{k}}: \left(\sum_{i \in S} \text{spHSS}_{|S|} \cdot \text{Eval}^{\text{sp}}(\text{rank}_S(i), \text{share}_{i,S}, (g_{S,j,\vec{k},\ell})_\ell) = \prod_{\ell=1}^{|S|} g_{S,j,\vec{k},\ell}(x^{(s_\ell)}) \right).$$

By correctness of $\text{spHSS}_{|S|}$ (whose party $\text{rank}_S(i)$ holds the semi-private input $x^{(i)}$), each $E_{S,j,\vec{k}}$ holds with probability $1 - \text{negl}(\lambda)$. There are at most n^d such S , r such j , and d^d such $\vec{k}$, all polynomial in λ by assumption, so by a union bound they all hold simultaneously with probability $1 - \text{negl}(\lambda)$; condition on this. By definition of the $g_{S,j,\vec{k},\ell}$, $\prod_{\ell} g_{S,j,\vec{k},\ell}(x^{(s_\ell)}) = \prod_{\ell} h_{s_\ell,j}(x^{(s_\ell)})^{k_\ell}$, so summing the definition of $w_{i,j}$ over $i \in [n]$ and applying Eq. (2) with $y_i \leftarrow h_{i,j}(x^{(i)})$ gives, for every j with $d_j \geq 1$,

$$\begin{aligned} \sum_{i \in [n]} w_{i,j} &= \left(\sum_{i=1}^n h_{i,j}(x^{(i)}) \right)^{d_j} \\ &= \left[\sum_{i=1}^{n-1} \left(f_j(x^{(i)}) - f_j(0_{\mathcal{R}^m}) \right) + f_j(x^{(n)}) \right]^{d_j} = f_j(x)^{d_j}, \end{aligned}$$

where the last equality uses that f_j is affine and $\sum_i x^{(i)} = x$. For j with $d_j = 0$, $\sum_i w_{i,j} = 1 = f_j(x)^0$ by construction. Hence $\sum_{i \in [n]} y_i = \sum_j \alpha_j f_j(x)^{d_j} = f(x)$ with all but negligible probability.

- **Security.** This is analogous to the security argument of Theorem 16. Let $\mathcal{Z} \subsetneq [n]$ be the set of corrupted parties; since some party is honest, the additive shares $(x^{(i)})_{i \in \mathcal{Z}}$ are uniform in $(\mathcal{R}^m)^{\mathcal{Z}}$ and independent of x , so handing $x^{(i)}$ to party i reveals nothing about x (and nothing beyond what semi-privacy of the committees containing i already allows). The simulator samples $(x^{(i)})_{i \in \mathcal{Z}} \xleftarrow{\$} (\mathcal{R}^m)^{\mathcal{Z}}$; then, for every committee S with $S \cap \mathcal{Z} \neq \emptyset$, it runs the simulator $\text{Sim}_{|S|}^{\text{sp}}$ of $\text{spHSS}_{|S|}$ with respect to the corrupted parties $\text{rank}_S(S \cap \mathcal{Z})$, input sizes $(m, \dots, m)$, and inputs $(x^{(i)})_{i \in S \cap \mathcal{Z}}$, obtaining $(\text{share}_{i,S})_{i \in S \cap \mathcal{Z}}$; committees with $S \cap \mathcal{Z} = \emptyset$ contribute nothing to the view of the corrupted parties. A hybrid argument over the polynomially many committees, each step justified by semi-privacy of $\text{spHSS}_{|S|}$, shows that the output $(x^{(i)}, (\text{share}_{i,S})_{S \ni i})_{i \in \mathcal{Z}}$ of the simulator is computationally indistinguishable from the real world. $\square$