Homomorphic Secret Sharing for Low Degree Polynomials

Russell W. F. Lai, Giulio Malavolta, and Dominique Schröder

Friedrich-Alexander-Universität Erlangen-Nürnberg

Abstract. Homomorphic secret sharing (HSS) allows n clients to secret-share data to m servers, who can then homomorphically evaluate public functions over the shares. A natural application is outsourced computation over private data. In this work, we present the first plain-model homomorphic secret sharing scheme that supports the evaluation of polynomials with degree higher than 2. Our construction relies on any degree-k (multi-key) homomorphic encryption scheme and can evaluate degree-$((k+1)m-1)$ polynomials, for any polynomial number of inputs n and any sub-logarithmic (in the security parameter) number of servers m. At the heart of our work is a series of combinatorial arguments on how a polynomial can be split into several low-degree polynomials over the shares of the inputs, which we believe is of independent interest.

1 Introduction

Homomorphic secret sharing (HSS), introduced by Boyle, Gilboa, and Ishai [9], allows n clients to secret share the data x into the shares $x_1, \ldots, x_m$ which are distributed to m servers, who can then homomorphically evaluate public functions over the shares. The evaluation is done locally by each server, meaning that there exists a local evaluation algorithm $\mathsf{Eval}(f, x_j)$ that takes as input a description of the function f and a share x_j, and returns a value y_j. The result of the distributed computation can be re-constructed using the decoding algorithm $\mathsf{Dec}(y_1, \ldots, y_m)$, which returns the result $f(x)$. HSS schemes for meaningful classes of functions can be constructed under weak assumptions, such as decisional Diffie-Hellman (DDH) [9] or the security of Paillier encryption scheme [20]. A natural application of HSS is outsourced computation over private data.

1.1 Our Contribution

We propose a family of HSS schemes for polynomials, and more generically for $(\sum\prod\sum)$-arithmetic circuits[1] from weak assumptions. More precisely, we show that:

Theorem 1 (Informal). *For all integers $k \geq 0$ and $m = O\left(\frac{\log\lambda}{\log\log\lambda}\right)$, if there exists a degree-k homomorphic public-key encryption scheme, then there exists an m-server homomorphic secret sharing for polynomials of degree $d = (k+1)m - 1$.*

[1] A $(\sum\prod\sum)$-arithmetic circuit [23] is a depth-3 circuit consisting of sum and product gates of unbounded fan-in, where the gates in the input layer and output layer consist of only sum gates, and those in the intermediate layer consists of only product gates.

Table 1: Comparison amongst existing HSS schemes for n clients, m servers and resilient against the corruption of t servers. "$n = *$" denotes unbounded number of clients.

	(n, m, t)	Functions	Assumptions
Shamir [30]	$(*, m, t)$	$\mathcal{R}_d$, $d = m - 1$	-
Benaloh [6]	$(*, m, m - 1)$	Affine	-
Information Theoretic PIR [31,18]	$(*, m, 1)$	Selection	-
Beimel et al. [5]	$(1, 3, 1)$	Depth 2 Boolean circuits	-
Computational PIR [13]	$(*, m, 1)$	Selection	Φ-Hiding
Function Secret Sharing [8,25]	$(1, m, m - 1)$	Point Function	OWF
Spooky Encryption [17]	$(*, m, m - 1)$	Circuits	LWE
Boyle et al. [9,10]	$(*, 2, 1)$	Branching Programs	DDH
Catalano and Fiore [15]	$(*, 2, 1)$	$\mathcal{R}_d$, $d = 2k$	k-HE
Section 4 and Section 5	$(*, m, 1)$	$\mathcal{R}_d$, $d = (k + 1)m - 1$	k-HE

Our scheme is perfectly correct, assuming a perfectly correct homomorphic encryption scheme, and naturally generalizes to the multi-key and the threshold settings. Our construction is secure in the plain model, without the need for a public-key setup. Interestingly, when $k = 0$, i.e., the encryption scheme has no homomorphic properties, we recover the same functionality of Shamir secret sharing [30], *i.e.*, the supported degree is $d = m - 1$. A comparison amongst existing HSS schemes is summarized in Table 1. Most of the known schemes are either limited to very restricted classes of functions (such as affine or point functions) or require assumptions from which we can instantiate fully homomorphic encryption (FHE), such as the learning with errors (LWE) assumption. Notable exceptions include the work of Catalano and Fiore [15] and the recent breakthrough result of Boyle, Gilboa, and Ishai [9]. The construction of Catalano and Fiore allows to efficiently outsource the computation of degree-$2k$ polynomials to 2 non-colluding servers, using only a degree-k homomorphic encryption scheme. Boyle et al. [9] proposed the first 2-server HSS scheme for branching programs (a superclass of NC1) assuming only the hardness of DDH. A shortcoming of this construction is that the correct result of the evaluation is recovered only with probability $\frac{1}{\mathsf{poly}(\lambda)}$. Additionally, their multi-key variant assumes the existence of a public-key setup.

Our result directly improves over the work of Catalano and Fiore [15] in two ways. First, we increase the computable degree d from $d = 2k$ to $2k + 1$. While this improvement seems small, it has significant consequences for small values of k: For example, for $k = 1$, we obtain a 2-server HSS for degree-3 polynomials, which can be bootstrapped to securely evaluate any function in P/poly [1] (assuming the existence of a PRG computable in NC1). In particular, it was shown [1] that any computation in P/poly can be probabilistically encoded by evaluating a set of polynomials of degree 3. The encoding can then be decoded in time proportional to the time complexity of the original computation. Furthermore, the encoding leaks nothing beyond the computation result.

Second, we generalize the scheme of Catalano and Fiore [15] for $m \geq 2$ servers, for any m which is sub-logarithmic in the security parameter. Increasing the number of supported servers allows us to relax the non-collusion assumption. We derive bounds for

the maximum degree supported by the resulting scheme and characterize the requirements for determining the minimum number of servers needed for correct computation.

1.2 Applications

Our HSS can be applied directly to outsource the computation of low-degree polynomials on private data. Examples of particular interests include:

1. Privacy-preserving machine learning using shallow neural networks where highly non-linear activation functions are approximated by low-degree (*e.g.*, degree-6 [24]) polynomials to be evaluated on private data.
2. Computation of several statistical measures over private data, such as variance, skewness and higher moments.
3. A round-optimal m-server PIR which can be casted as the evaluation of the selection function (a degree d polynomial) over a private index and the entire database DB for a communication complexity dominated by a factor $\frac{|\mathsf{DB}|}{2^d} + \mathsf{poly}(\lambda)$.

A recent work from Boyle et al. [11] describes how to generically bootstrap an additive 3-client 2-server HSS scheme for degree-3 polynomials (in the PKI model) into a round-optimal n-clients m-servers MPC protocol (in the PKI model) for any choice of n and m which are polynomial in the security parameter. Applying a similar transformation to our scheme we obtain a round-optimal n-clients m-servers *server-aided* MPC protocol in the plain model. Server-aided MPC [27,28,15] models real-life scenarios where clients outsource the burden of the computation to (non-colluding) cloud servers. In particular, such model allows the adversary to corrupt any strict subset of the servers *or* the output client, and an arbitrary number of input clients. Beyond being round-optimal, our MPC protocol has several interesting properties. First, it can be instantiated from any multi-key linearly homomorphic encryption (which can be constructed from the DDH assumption [14]). Additionally, our HSS scheme is perfectly correct and thus the transformation from HSS to MPC does not need to go through the probability amplification step of [10]. It also inherits the efficiency features of the transformation of Boyle et al. [11]: If $|f|$ is the size of the circuit computing f, then

- the computational efficiency is $|f| \cdot \mathsf{poly}(\lambda) \cdot n^3$ when $m = O(1)$ or $m = n$, and
- the output client complexity is bounded by $|f| \cdot \mathsf{poly}(\lambda) \cdot m$.

1.3 Our Techniques

We illustrate the basic ideas behind our HSS scheme with a simple example, where two servers wish to privately compute the function $f(x, y, z) = xyz$, for some values x, y, and z belonging to a ring R. The client computes a standard 2-out-of-2 Shamir's secret sharing of each input and arranges the shares into the following matrix:

$$T := \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \\ z_1 & z_2 \end{bmatrix} \; s.t. \; T \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

In the following, boxed shares are encrypted shares under a linearly homomorphic encryption scheme HE (assume for the moment under the same public key). The client then distributes the shares to the two servers $(\mathcal{S}_1, \mathcal{S}_2)$ as follows:

$$\mathcal{S}_1 \leftarrow \begin{bmatrix} \boxed{x_1} & x_2 \\ \boxed{y_1} & y_2 \\ \boxed{z_1} & z_2 \end{bmatrix} \qquad \mathcal{S}_2 \leftarrow \begin{bmatrix} x_1 & \boxed{x_2} \\ y_1 & \boxed{y_2} \\ z_1 & \boxed{z_2} \end{bmatrix}$$

It is not hard to see that distributing the shares in that way does not reveal any information to either of the servers. This follows from the the semantic security of the encryption scheme because each server alone cannot recover the plain value of the original inputs. Now, we show how the two servers can jointly compute the function f over the inputs x, y, and z. Let us expand the product

$$xyz = (x_1 + x_2)(y_1 + y_2)(z_1 + z_2) = \sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{\ell=1}^{2} x_i y_j z_\ell.$$

We now consider each term $x_i y_j z_\ell$ individually. By the pigeonhole principle, for each combination of indices $(i, j, \ell) \in \{1, 2\}^3$ there exists at least one server for which *at most* one of the entries is encrypted. As an example, $(1, 1, 2)$ is a "valid" set of indices for $\mathcal{S}_2$, since it knows the plain values x_1 and y_1 and the encrypted share $\mathsf{HE.Enc}(\mathsf{pk}, z_2)$. This implies that every monomial is computable by a server by treating the plaintext entries as a constant and multiplying them to the encrypted entry, e.g.,

$$\mathsf{HE.Enc}(\mathsf{pk}, z_2)^{x_1 \cdot y_1} = \mathsf{HE.Enc}(\mathsf{pk}, x_1 y_1 z_2).$$

This kind of operations is supported by the encryption scheme since it is linearly homomorphic. Let $\mathcal{I}_1 \subset \{1, 2\}^3$ be the set of valid indices for the server $\mathcal{S}_1$, let $\mathcal{I}_2 := \{1, 2\}^3 \setminus \mathcal{I}_1$ be the set for $\mathcal{S}_2$, and let m_i be the monomial indexed by the i-th set of indices. Exploiting the homomorphic properties of the encryption scheme each server computes

$$\mathsf{HE.Enc}\left(\mathsf{pk}, \sum_{i \in \mathcal{I}_1} m_i\right) \leftarrow \mathcal{S}_1 \qquad \mathsf{HE.Enc}\left(\mathsf{pk}, \sum_{i \in \mathcal{I}_2} m_i\right) \leftarrow \mathcal{S}_2$$

and sends the two ciphertext to the client. The client (who knows the secret key) can decrypt and sum the plaintexts up to recover the result of the computation: $\sum_{i \in \mathcal{I}_1} m_i + \sum_{i \in \mathcal{I}_2} m_i = \sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{\ell=1}^{2} x_i y_j z_\ell = xyz$. Although the two plaintexts may contain some information of the intermediate values of the computation, this can be easily avoided by adding a dummy sum of two shares of 0. This immediately extends to the computation of any degree-3 polynomial.

Increasing the degree of the polynomial. The next observation is that, using the same principle, increasing the number of servers also increases the degree of the polynomial that can be computed. The inputs are shared across m servers with the same strategy and

the view of each server looks as follows:

$$S_j \leftarrow \begin{bmatrix} x_1 & \dots & x_{j-1} & \boxed{x_j} & x_{j+1} & \dots & x_m \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ z_1 & \dots & z_{j-1} & \boxed{z_j} & z_{j+1} & \dots & z_m \end{bmatrix}$$

Products are computed as before and a simple combinatorial argument shows that the maximum degree computable is $d = 2m-1$. Extending to an arbitrary amount of servers introduces some subtlety in the splitting of monomials since now one combination of indices might be computable by more than one server. Thus one needs to take some extra care in the design of a suitable splitting function.

Furthermore, if we admit the existence of a homomorphic encryption scheme for degree-k polynomials, then the degree computable by each server increases even more since now k encrypted entry can be multiplied together locally. Our analysis shows that the degree increases to $d = (k+1)m - 1$.

Extensions. We consider some natural extensions of our HSS scheme. In a multi-key HSS, clients can independently share their inputs such that servers can evaluate functions over an arbitrary set of shares. Since the shares from different clients are tied to different public keys, we need to upgrade the baseline homomorphic encryption scheme to a multi-key homomorphic encryption scheme. For completeness, we also explore the feasibility of increasing the corruption threshold t by increasing the amount of encrypted entries per server (and decreasing the maximum supported degree d).

1.4 Related Work

Similar techniques on splitting the evaluation of polynomials have been used in the context of simultaneous-message multiparty computation [2] and private information retrieval [4]. Barkol *et al.* [3] leveraged a similar observation to prove an upper bound on the degree of polynomials computable by any information theoretic secret sharing scheme. Another closely related work is by Franklin and Mohassel [21], who propose a two party computation protocol for degree-3 polynomials. However, their protocol is interactive and therefore does not imply a homomorphic secret sharing scheme.

2 Preliminaries

Notations. We denote by $\lambda \in \mathbb{N}$ the security parameter and by $\mathsf{poly}(\lambda)$ any function that is bounded by a polynomial in λ. We address any function that is *negligible* in the security parameter with $\mathsf{negl}(\lambda)$. An algorithm is PPT if it is modeled as a probabilistic Turing machine whose running time is bounded by some function $\mathsf{poly}(\lambda)$. Given a set S, we denote by $x \leftarrow S$ the sampling of an element uniformly at random in S and by $[n]$ we denote the set of integers $\{1, \dots, n\}$. In the following we recall the definition of statistical distance.

Definition 1 (Statistical Distance). *Let X and Y be two random variables over a finite set $\mathcal{U}$. The statistical distance between X and Y is defined as*

$$\mathbb{SD}[X, Y] = \frac{1}{2} \sum_{u \in \mathcal{U}} |\Pr[X = u] - \Pr[Y = u]| .$$

2.1 Homomorphic Encryption

For conciseness, in the remaining of this section, we work with multivariate polynomials with the number of variables fixed to n. All results can be generalized to the case with unbounded number of variables. Formally, let R be a (finite) ring and $\mathcal{R} := R[X_1, \ldots, X_n]$ be the ring of n-variate polynomials over R. Let $\mathcal{R}_d := \{f \in \mathcal{R} : \deg(f) \leq d\}$ be a set of such polynomials of degree at most d. We recall the notion of homomorphic encryption, for which we assume that the message domain $\mathcal{M}$ of the scheme is a finite ring R that is publicly known and where it is possible to efficiently sample uniformly distributed elements (*e.g.*, [7,14,29], see [15] for a more comprehensive list).

Definition 2 (Homomorphic Encryption (HE)). *A* public key homomorphic encryption scheme $\mathsf{HE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *over degree-d polynomials $\mathcal{R}_d$, consists of the following* PPT *algorithms:*

$\mathsf{KGen}(1^\lambda)$: *The* key generation *algorithm takes as input the security parameter λ and outputs the public key* pk *and the secret key* sk.

$\mathsf{Enc}(\mathsf{pk}, m)$: *The* encryption *algorithm takes as input the public key* pk *and the message $m \in \mathcal{M}$; it returns a ciphertext $c \in \mathcal{C}$.*

$\mathsf{Eval}(\mathsf{pk}, f, (c_1, \ldots, c_n))$: *The* evaluation *algorithm takes as input the public key* pk*, a polynomial $f \in \mathcal{R}_d$, and a vector of n ciphertexts $(c_1, \ldots, c_n) \in \mathcal{C}^n$; it returns a ciphertext $c \in \mathcal{C}$.*

$\mathsf{Dec}(\mathsf{sk}, c)$: *The* decryption *algorithm takes as input the private key* sk *and a ciphertext $c \in \mathcal{C}$; it returns a plaintext $m \in \mathcal{M}$.*

Correctness. A homomorphic encryption scheme has *decryption correctness* if for any $\lambda \in \mathbb{N}$, any $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KGen}(1^\lambda)$, and any message $m \in \mathcal{M}$, we have that

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m] \geq 1 - \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of Enc.

A homomorphic encryption scheme has *(2-hop) evaluation correctness* if for any $\lambda \in \mathbb{N}$, any $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KGen}(1^\lambda)$, any polynomials $f, f_1, \ldots, f_n \in \mathcal{R}$ such that $f(f_1, \ldots, f_n) \in \mathcal{R}_d$, any messages $m, m_i \in \mathcal{M}$ for $i \in [n]$ where $m = f(f_1(m_1, \ldots, m_n), \ldots, f_n(m_1, \ldots, m_n))$, we have that

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, c) = m : \begin{array}{c} \forall i \in [n], c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, m_i), \\ c \leftarrow \mathsf{Eval}(\mathsf{pk}, f, (c_1, \ldots, c_n)) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of Enc and Eval. The scheme is *perfectly correct* if the above probabilities are exactly 1.

Compactness. We sometimes require a homomorphic encryption scheme to be *compact*. This imposes a bound on the size of the output of Eval: The size of the output (and consequently the running time of Dec) must be independent from the size of the evaluated polynomial (*e.g.*, when expressed as a circuit) [22].

Security. The security of a homomorphic encryption scheme is the standard notion of semantic security introduced by Goldwasser and Micali [26].

Definition 3 (Semantic Security). *A homomorphic encryption scheme* HE *is* IND-CPA*-secure (has indistinguishable messages under chosen plaintext attack) if for any* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\mathsf{Pr}\left[b = b' : \begin{array}{c}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda), (m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}_1(\mathsf{pk}),\\ b \leftarrow \{0,1\}, c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b), b' \leftarrow \mathcal{A}_1(\mathsf{state}, c)\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of b, KGen, *and* Enc.

Circuit Privacy. In the context of homomorphic encryption, semantic security might be per se not sufficient to guarantee the secrecy of the encrypted messages. In particular, the output of Eval may still contain some information about the messages encrypted in the input ciphertexts. This leakage is ruled out by the notion of *circuit privacy* [12].

Definition 4 (Circuit Privacy). *A homomorphic encryption scheme* HE *is* circuit-private *with respect to a family of functions* $\mathcal{F}$*, if there exists a* PPT *simulator* $\mathcal{S}_{\mathsf{HE}}$ *and a negligible function* $\mathsf{negl}(\lambda)$ *such that for any* $\lambda \in \mathbb{N}$*, any* $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KGen}(1^\lambda)$*, any* $f \in \mathcal{F}$*, any vector of messages* $(m_1, \ldots m_n) \in \mathcal{M}^n$*, and any vector of ciphertexts* $(c_1, \ldots, c_n) \in \mathcal{C}^n$ *such that for all* $i \in \{1 \ldots t\} : c_i \in \mathsf{Enc}(\mathsf{pk}, m_i)$*, we have that*

$$\mathbb{SD}\left[\mathsf{Eval}(\mathsf{pk}, f, (c_1, \ldots, c_n)), \mathcal{S}_{\mathsf{HE}}(1^\lambda, \mathsf{pk}, f(m_1, \ldots, m_n))\right] \leq \mathsf{negl}(\lambda)\,.$$

Multi-Key Homomorphic Encryption. The above definition of homomorphic encryption can be extended to the multi-client settings with minimal changes. To do so, we consider the scenario of n clients, each holding an independent key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$: The key generation and encryption algorithms are unchanged whereas the evaluation and the decryption algorithms take as input vectors of public and secret keys, respectively, and are defined as $\mathsf{Eval}((\mathsf{pk}_1, \ldots, \mathsf{pk}_n), f, (c_1, \ldots, c_n))$ and $\mathsf{Dec}((\mathsf{sk}_1, \ldots, \mathsf{sk}_n), c)$. The definitions of correctness and circuit privacy can easily be modified accordingly.

The (lifted) ElGamal encryption scheme [19] is an example of multi-key homomorphic encryption. Informally, given the ciphertexts $(g^r, h_1^r g^{m_1})$ and $(g^s, h_2^s g^{m_2})$ of m_1 and m_2 under the keys $h_1 = g^{x_1}$ and $h_2 = g^{x_2}$ respectively, one can compute $(g^r, g^s, h_1^r h_2^s g^{m_1+m_2})$ as a ciphertext of $m_1 + m_2$ under the combined key (h_1, h_2). The decryption of lifted ElGamal requires the computation of a discrete logarithm and therefore it is important that the evaluated message lies in a polynomial space. To overcome this limitation, one can use the variant of Castagnos and Laguillaumie [14].

3 Definition of Homomorphic Secret Sharing

We define a variant of homomorphic secret sharing [9] in the public-key setup model. Our variant considers three parties: one output client, many input clients, and many servers. The output client provides the setup, meaning that it generates a public and a secret key and it shares the public key among all participants. Furthermore, it also computes the final result. The input clients secret share their inputs as "input shares" to all servers. The servers homomorphically evaluate functions, in our case polynomials, over the input shares to obtain "output shares". These output shares are then sent to the output client, who uses its secret key to decode them.

The definition can be generalized to the multi-output- client (or multi-key) setting, where different parties receive the output of the computation. The definition can also be lifted to the plain model by simply removing the key generation algorithm and letting all public and secret keys inputs be empty strings.

Definition 5 (Homomorphic Secret Sharing (HSS)). *An* n-input (1-output) m-server homomorphic secret sharing *scheme for degree-*d *polynomials* $\mathcal{R}_d$ *(with public-key setup)* $\mathsf{HSS} = (\mathsf{KGen}, \mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ *consists of the following* PPT *algorithms / protocols:*

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$: *On input the security parameter* 1^λ*, the* key generation *algorithm outputs a public key* pk *and a secret key* sk*.*

$(s_{i,1}, \ldots, s_{i,m}) \leftarrow \mathsf{Share}(\mathsf{pk}, i, x)$: *Given a public key* pk*, an input index* $i \in [n]$*, and an input* $x \in R$*, the* sharing *algorithm outputs a set of shares* $(s_{i,1}, \ldots, s_{i,m})$*.*

$y_j \leftarrow \mathsf{Eval}(\mathsf{pk}, j, f, \{s_{i,j}\}_{i\in[n]})$: *The* evaluation *protocol is executed by a server* $\mathcal{S}_j$ *on inputs the public key* pk*, an index* j*, a degree-*d *polynomial* f*, and the corresponding tuple of shares* $(s_{i,j})_{i\in[n]}$*. Upon termination, the server* $\mathcal{S}_j$ *outputs the corresponding output share* y_j*.*

$y \leftarrow \mathsf{Dec}(\mathsf{sk}, (y_1, \ldots, y_m))$: *On input a secret key* sk *and a tuple of output shares* $(y_1, \ldots, y_m)$*, the* decoding *algorithm outputs the result* y *of the evaluation.*

Correctness. An n-input m-server HSS scheme for degree-d polynomials $\mathcal{R}_d$ is *correct* if for any $\lambda \in \mathbb{N}$, any $m, n \in \mathsf{poly}(\lambda)$, any $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KGen}(1^\lambda)$, any $f \in \mathcal{R}_d$, any n-tuple of inputs $(x_1, \ldots, x_n) \in R^n$, it holds that

$$\mathsf{Pr}\left[\begin{array}{l}\mathsf{Dec}(\mathsf{sk}, (y_1, \ldots, y_m)) = f(x_1, \ldots, x_n) : \\ \quad \forall i \in [n], (s_{i,1}, \ldots, s_{i,m}) \in \mathsf{Share}(\mathsf{pk}, i, x_i), \\ \quad \forall j \in [m], y_j \in \mathsf{Eval}(\mathsf{pk}, j, f, \{s_{i,j}\}_{i\in[n]})\end{array}\right] \geq 1 - \mathsf{negl}(\lambda)\,,$$

where the probability is taken over the random coins of Share and Eval. The scheme is *perfectly correct* if the above probability is exactly 1.

Security. The security of a HSS scheme guarantees that no information about the message is disclosed to any subset of servers of size at most t.

Definition 6 (Security). *An* n*-input* m*-server HSS scheme is* t-secure *if for any* $\lambda \in \mathbb{N}$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for any* PPT *algorithm* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*,*

$$\left|\mathsf{Pr}\left[\mathsf{Security}^0_{\mathcal{A},\mathsf{HSS}} = 1\right] - \mathsf{Pr}\left[\mathsf{Security}^1_{\mathcal{A},\mathsf{HSS}} = 1\right]\right| < \mathsf{negl}(\lambda)$$

$\mathsf{Security}^{b}_{\mathcal{A},\mathsf{HSS}}(1^\lambda)$:	$\mathsf{Context\text{-}Hiding}^{b}_{\mathcal{A},\mathcal{S},\mathsf{HSS}}(1^\lambda)$:
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$	$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
$(x_0, x_1, J, \mathsf{st}) \leftarrow \mathcal{A}_0(\mathsf{pk})$	$(f, x_1, \ldots, x_n, \mathsf{st}) \leftarrow \mathcal{A}_0(\mathsf{pk})$
$(s_1, \ldots, s_m) \leftarrow \mathsf{Share}(\mathsf{pk}, x_b)$	**if** $b = 0$ **then**
$b' \leftarrow \mathcal{A}_1(\mathsf{st}, \{s_j : j \in J\})$	$(s_{i,1}, \ldots, s_{i,m}) \leftarrow \mathsf{Share}(\mathsf{pk}, i, x_i)\ \forall i \in [n]$
$b_0 := (J \subseteq \mathbb{N} \wedge \lvert J\rvert \leq t)$	$y_j \leftarrow \mathsf{Eval}(\mathsf{pk}, j, f, \{s_{i,j}\}_{i\in[n]})\ \forall j \in [m]$
$b_1 := (b = b')$	**elseif** $b = 1$ **then**
return $b_0 \wedge b_1$	$(y_1, \ldots, y_m) \leftarrow \mathcal{S}(1^\lambda, \mathsf{pk}, f(x_1, \ldots, x_n))$
	endif
	$b' \leftarrow \mathcal{A}_1(\mathsf{st}, (y_1, \ldots, y_m))$
	return 1 iff $b' = b$

Fig. 1: Security experiments for $(*, m, t)$-HSS

where $\mathsf{Security}^{b}_{\mathcal{A},\mathsf{HSS}}$ *is defined in Figure 1 for* $b \in \{0, 1\}$.

For conciseness, we refer to an n-input, m-server, t-secure homomorphic secret sharing scheme as an (n, m, t)-HSS. If the number of inputs is unbounded, we denote it by $(*, m, t)$-HSS.

Robustness. An (n, m, t)-HSS scheme is r*-robust* if it suffices for the output client to collect output shares from any r out of m servers to recover the computation result.

Context Hiding. In the setting of outsourced computations, the party who decrypts may be different from the one who provides the inputs of the computation or determines the function to be computed. For this reason, Catalano and Fiore [15] introduced the notion of context-hiding, which assures that the decrypting party learns nothing beyond the output of the computation.

Definition 7 (Context Hiding). *A* (n, m, t)*-HSS scheme is context-hiding if for any* $\lambda \in \mathbb{N}$ *there exists a* PPT *simulator* $\mathcal{S}$ *and a negligible function* $\mathsf{negl}(\lambda)$ *such that for any* PPT *algorithm* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*,*

$$\left|\mathsf{Pr}\left[\mathsf{Context\text{-}Hiding}^{b}_{\mathcal{A},\mathcal{S},\mathsf{HSS}} = 1\right] - \mathsf{Pr}\left[\mathsf{Context\text{-}Hiding}^{b}_{\mathcal{A},\mathcal{S},\mathsf{HSS}} = 1\right]\right| < \mathsf{negl}(\lambda)$$

where $\mathsf{Context\text{-}Hiding}^{b}_{\mathcal{A},\mathcal{S},\mathsf{HSS}}$ *is defined in Figure 1 for* $b \in \{0, 1\}$.

Multi-Key HSS. Homomorphic secret sharing can be easily generalized to the multi-key/multi-input-client settings by extending the evaluation protocol so that all servers take as input all public keys of the participating output clients, and the decryption algorithm takes as input all of the corresponding secret keys. While the definition of security is unchanged and is required to hold for each secret key, the definitions of correctness, robustness, and context-hiding are extended accordingly.

4 Main Construction in the Public-Key Model

Let m and k be positive integers. We present a generic construction of an unbounded-input (1-output) m-server 1-secure homomorphic secret sharing ($(*, m, 1)$-HSS) scheme HSS for degree-d polynomials $\mathcal{R}_d$ in the public-key model, where $d = (k+1)m-1$. Our construction is generic and relies only a public key homomorphic encryption scheme HE for degree-k polynomials. We analyze the efficiency of our construction in Section 4.3 and show that it satisfies the security definitions for an HSS scheme in Section 4.4. For the sake of simplicity, we initially assume a public-key setup and we show how to upgrade it to the plain model in Section 5.

4.1 Construction

In the following we provide the reader with an intuitive description of our main construction and we refer to Figure 2 for a formal description.

Key Generation. On input the security parameter, the output client generates the keys of the encryption HE scheme and publishes the public key.

Secret Sharing. To secret share a ring element $x_i \in R$, the input client samples random base secret shares $x_{i,j} \leftarrow R$ for $j \in [m]$ subject to the constraint that $\sum_{j\in[m]} x_{i,j} = x_i$. It then encrypts each share $x_{i,j}$ for $i \in [n]$, $j \in [m]$ as $\tilde{x}_{i,j}$. Similarly, base shares of 0 are randomly sampled as $(z_{i,1}, \ldots, z_{i,m}) \in R^m$ such that $\sum_{j\in[m]} z_{i,j} = 0$. For each $j' \in [m]$, the resulting j'-th secret share of $(x_1, \ldots, x_n)$ consists of all plaintext base shares $x_{i,j}$ for all $i \in [n]$ and $j \in [m] \setminus \{j'\}$, the encrypted base shares $\tilde{x}_{i,j'}$ for all $i \in [n]$, and the plain 0-shares $z_{i,j'}$ for all $i \in [n]$. The process of creating a share $s_{i,j'}$ is visualized below and formalized in Figure 2.

$$
\begin{aligned}
0 &\xrightarrow{\text{Base Share}} \left[\, z_{i,1} \cdots z_{i,m} \,\right] \\
x_i &\xrightarrow{\text{Base Share}} \left[\, x_{i,1} \cdots x_{i,j'-1} \;\; x_{i,j'} \;\; x_{i,j'+1} \cdots x_{i,m} \,\right] \\
&\xrightarrow{\mathsf{HE.Enc}} \left[\, x_{i,1} \cdots x_{i,j'-1} \; \boxed{\tilde{x}_{i,j'}} \; x_{i,j'+1} \cdots x_{i,m} \; z_{i,j'} \,\right] := s_{i,j'}
\end{aligned}
$$

Evaluation. Let $f \in \mathcal{R}_d$ be an n-variate polynomial of degree at most d for some $n \in \mathsf{poly}(\lambda)$. Without loss of generality, suppose the servers are to homomorphically evaluate f over x_i for $i \in [n]$ which have been secret shared as $(s_{i,1}, \ldots, s_{i,m})$ respectively[2]. To do so, $\mathcal{S}_j$ locally evaluates a function f_j over its shares $(s_{1,j}, \ldots, s_{n,j})$, to be explained below. We will construct (for each d) a function Split_d that splits f into some polynomials $g_1, \ldots, g_m$, with the following properties:

1. If $\sum_{j\in[m]} x_{i,j} = x_i$, then $f(x_1, \ldots, x_n) = \sum_{j\in[m]} g_j(x_{1,j}, \ldots, x_{n,j})$.

[2] In general, i can be picked from any index set $I \subset \mathbb{N}$ of size n.

2. For all $i \in [n]$ and $j \in [m]$, fix $x_{i,j}$ such that $\sum_{j\in[m]} x_{i,j} = x_i$. Let $x_i^{-j} = (x_{i,1}, \ldots, x_{i,j-1}, x_{i,j+1}, \ldots, x_{i,m})$. Then for each $j \in [m]$, g_j is an n-variate polynomial over $(x_{1,j}, \ldots, x_{n,j})$ of degree at most k, whose coefficients are uniquely determined by f and $(x_1^{-j}, \ldots, x_n^{-j})$, denoted by $g_j = \mathsf{Split}_d(j, f, (x_1^{-j}, \ldots, x_n^{-j}))$.

Note that f and $(x_1^{-j}, \ldots, x_n^{-j})$ are known by server $\mathcal{S}_j$ in plaintext. Since the underlying encryption scheme supports degree k polynomials, $\mathcal{S}_j$ can evaluate $f_j = g_j + \sum_{i\in[n]} z_{i,j}$ over the encrypted base shares $(\tilde{x}_{1,j}, \ldots, \tilde{x}_{n,j})$ to obtain the output share y_j, which is a ciphertext encrypting $g_j(x_{1,j}, \ldots, x_{n,j}) + \sum_{i\in[n]} z_{i,j}$.

It remains to show how to construct the Split_d function. Let $f = f(X_1, \ldots, X_n)$ be a polynomial of degree d, write $f = \sum_w a_w M_w(X_1, \ldots, X_n)$, where M_w are monomials of some degree $c \le d$, where c depends on w, with coefficients a_w. For each w, consider the monomial $M_w(X_1, \ldots, X_n) = X_{w_1} X_{w_2} \cdots X_{w_c}$ for some (possibly duplicating) indices $w_1, \ldots, w_c \in [n]$. Next, by defining a set of new variables $X_{i,j}$ for all $i \in [n]$ and $j \in [m]$ and substituting $X_i = \sum_{j\in[m]} X_{i,j}$, we can expand the monomial M_w as

$$\begin{aligned} M_w(X_1, \ldots, X_n) &= X_{w_1} X_{w_2} \cdots X_{w_c} \\ &= \prod_{i\in\{w_1,\ldots,w_c\}} \left(\sum_{j=1}^{m} X_{i,j} \right) \\ &= X_{w_1,1} \cdots X_{w_c,1} + \ldots + X_{w_1,m} \cdots X_{w_c,m} \\ &= \sum_{e\in[m]^c} X_{w_1,e_1} \cdots X_{w_c,e_c}. \end{aligned}$$

We now inspect the summand $X_{w_1,e_1} \cdots X_{w_c,e_c}$. Recall that $c \le d = (k+1)m - 1$. By the (dual of) pigeonhole principle, any way of writing c as a sum of m non-negative integers must contain a summand which is at most k, where the "worst case" is $c = \underbrace{(k+1) + \ldots + (k+1)}_{m-1} + k$. In other words, for each summand $X_{w_1,e_1} \cdots X_{w_c,e_c}$, there exists an index $j \in \{e_1, \ldots, e_c\}$ which appears at most k times in the expression. Furthermore, such an index j can be chosen deterministically by a publicly known algorithm. For the moment, we will continue the description of the construction without specifying explicitly such an algorithm. In Section 4.3 we will give two explicit examples and analyze their efficiency.

With this observation in mind, we can rewrite each monomial M_w as

$$\begin{aligned} M_w(X_1, \ldots, X_n) &= \sum_{e\in[m]^c} X_{w_1,e_1} \cdots X_{w_c,e_c} \\ &= \sum_{j\in[m]} h_{w,j}[X_1^{-j}, \ldots, X_n^{-j}](X_{1,j}, \ldots, X_{n,j}) \end{aligned}$$

where $X_i^{-j} = (X_{i,1}, \ldots, X_{i,j-1}, X_{i,j+1}, \ldots, X_{i,n})$ is defined similar to x_i^{-j}, and each term $h_{w,j}[X_1^{-j}, \ldots, X_n^{-j}](X_{1,j}, \ldots, X_{n,j})$ is the sum over the subset of all summands with j being the chosen index as defined by the property above. Note that each $h_{w,j}$ can

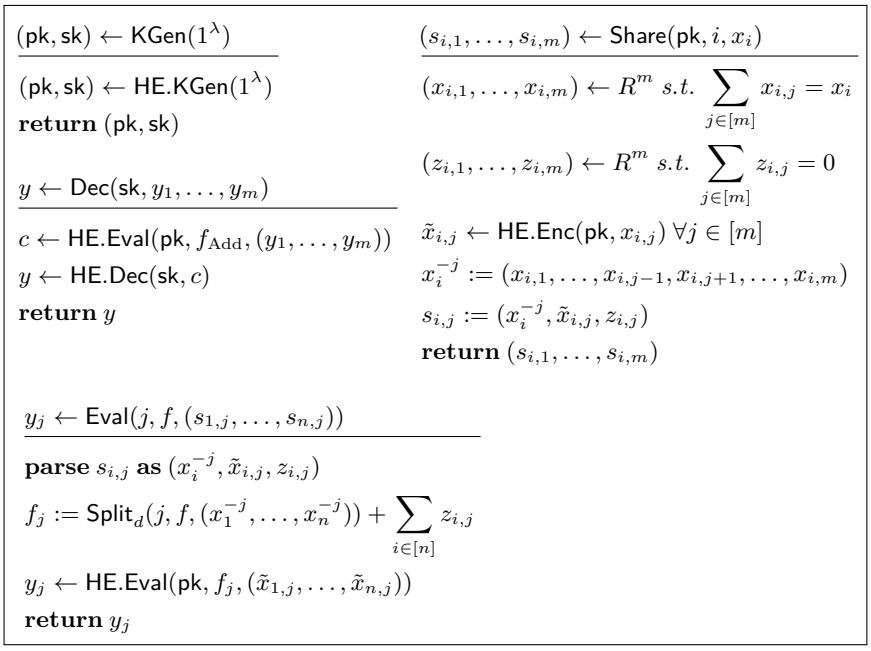

Fig. 2: Construction of a homomorphic secret sharing scheme HSS in the public-key setup model. (The functions f_{Add} and Split_d are defined in the text description.)

be interpreted as a degree-k (at most) polynomial over $X_{1,j}, \ldots, X_{n,j}$, with coefficients depending on $X_1^{-j}, \ldots, X_n^{-j}$.

Finally, we can rewrite the polynomial $f(X_1, \ldots, X_n)$ as

$$f(X_1, \ldots, X_n) = \sum_w a_w M_w = \sum_w a_w \sum_{j\in[n]} h_{w,j} = \sum_{j\in[m]} \sum_w a_w h_{w,j}.$$

Since each $h_{w,j}$ is a degree-k polynomial over $X_{1,j}, \ldots, X_{n,j}$, we can define $g_j := \sum_w a_w h_{w,j}$ which is a degree-k polynomial function over $X_{1,j}, \ldots, X_{n,j}$, with coefficients uniquely determined by f and $X_1^{-j}, \ldots, X_n^{-j}$. This completes the construction of the Split_d function.

Decoding. Let $f_{\mathrm{Add}}(Y_1, \ldots, Y_m) = Y_1 + \ldots + Y_m$. Since f_{Add} is of degree-1, its homomorphic evaluation is supported by HE. Thus, given the output shares $y_1, \ldots, y_m$ (which are HE ciphertexts), the output client can homomorphically evaluate f_{Add} over $y_1, \ldots, y_m$, which are ciphertexts encrypting $f_j(x_1^{-j}, \ldots, x_n^{-j})$ respectively. By the construction of f_j, the evaluation yields a ciphertext encrypting $f(x_1, \ldots, x_n)$, which can then be decrypted using sk.

Correctness. Note that by condition 2 of the Split_d algorithm the function g_j (and consequently f_j) is of degree at most k. Therefore the polynomial $f_{\mathrm{Add}}(f_1, \ldots, f_m)$ is

of degree at most k. By the evaluation correctness of HE, we have that

$$y = \mathsf{HE.Dec}(\mathsf{sk}, c) = \sum_{j\in[m]} g_j(x_{1,j}, \ldots, x_{n,j}) + \sum_{i\in[n]} \sum_{j\in[m]} z_{i,j}$$

except with negligible probability. By condition 1 of the Split_d algorithm and since for all $i \in [n]$ it holds that $\sum_{j\in[m]} z_{i,j} = 0$, we have that

$$\begin{aligned} y = & \sum_{j\in[m]} g_j(x_{1,j}, \ldots, x_{n,j}) + \sum_{i\in[n]} \sum_{j\in[m]} z_{i,j} \\ = & \ f(x_1, \ldots, x_n) + \sum_{i\in[n]} 0 = \ f(x_1, \ldots, x_n). \end{aligned}$$

Note that if HE is perfectly correct, then HSS is also perfectly correct.

Remark 1 (The case $k = 0$). For completeness, we remark that the construction still works for the case of $k = 0$ (*i.e.*, HE is a public-key *non-homomorphic* encryption) for the most part, except that "homomorphic evaluations" (in the evaluation protocol and the decoding algorithm) are performed over the plaintext shares rather than ciphertexts. The ciphertexts can actually be discarded or not created in the first place.

4.2 Discussion

In the following we first argue that our techniques are "tight" with respect to the degree of the polynomial to be computed. Then we discuss some function-dependent optimizations that we can apply to improve the efficiency of our protocol.

On the upper bound of the supported degree d. In our construction, we showed that it is possible to support the evaluation of polynomials of degree at most $(k + 1)m - 1$ using m servers. By a counting argument, we can show that $(k + 1)m - 1$ is also the maximum possible supported degree of our construction. Suppose the n servers were to evaluate a degree $(k + 1)m$ polynomial f, then f contains a monomial $M_w = \sum_{j_1,\ldots,j_{c_w}\in[m]} X_{i_{w,1},j_1} \cdots X_{i_{w,c_w},j_{c_w}}$ of degree $(k + 1)m$ which contains a summand $X_{i_{w,1},j_1} \cdots X_{i_{w,c_w},j_{c_w}}$ in which each $j \in \{j_1, \ldots, j_{c_w}\} = [m]$ appears exactly $k + 1$ times in the expression. Thus, it is impossible to write $M_w = \sum_{j\in[m]} h_{w,j}[X_{i_1}^{-j}, \ldots, X_{i_c}^{-j}](X_{i_{w,1},j}, \ldots, X_{i_{w,c_w},j})$ such that each $h_{w,j}$ is a polynomial of degree at most k. In fact, no matter how the indices j are chosen, there must exists $h_{w,j}$ which is of degree at least $k + 1$.

On computing polynomials with a subset of servers. A naïve usage of our HSS scheme requires one to query all of the m servers even when the degree of the polynomial is lower than d. In fact, a smaller number of servers could be used, with the following modification: Consider any subset $M \subseteq [m]$ and any ordering of its indices (say lexicographical). All base shares of each input $x_i = (x_{i,1}, \ldots, x_{i,m})$ for m servers can be transformed into base shares for any set of $|M|$ servers by setting

$$x_i \overset{\text{Base Share}}{\to} \left[x_{i,j^*} + \sum_{j\notin M} x_{i,j}, \{x_{i,j}\}_{j\in M\setminus j^*} \right]$$

where j^* is the first element of M. This operation can be computed also homomorphically (when needed). Note that the resulting base share is a well-formed input for the set M. Additionally one needs to provide each server with all of the 0-shares $(z_{i,1}, \ldots, z_{i,m})$ and apply the same transformation as described above. The resulting share $s_{i,j}$ is a correctly formed share for a set of $|M|$ servers. Note that this operation can be performed non-interactively by all servers belonging to M.

In another perspective, one can view this as a mechanism for performing the evaluation (although for a lower degree polynomial) using only a subset of the servers of size r, *i.e.*, a r-robust HSS scheme. Using combinatorial arguments similar to those in Section 4.1, one can conclude that any size-r subset of the m servers is able to evaluate polynomials of degree $km + r - 1$.

On arithmetic circuits of the form $\sum\prod\sum$. In the construction of the Split_d function, we assume that the polynomial f to be evaluated is given in the fully expanded form. In general, the number of monomials in a fully expanded polynomial of degree d is exponential in d. It is therefore desirable if the Split_d function can handle representations of polynomials f which are not fully unrolled. In certain special cases, this might save an exponential factor (of the number of monomials and hence server computation complexity) in the degree of the polynomial.

Our observation is that in our construction, computing linear functions over the inputs is essentially "free": Given a linear function $\mathcal{L}(X_1, \ldots, X_n)$, and a set of shares $\{s_{i,j}\}_{i=1}^n = \{(x_i^{-j}, \tilde{x}_{i,j}, z_{i,j})\}_{i=1}^n$, each server $\mathcal{S}_j$ can locally compute $x'_{j'} = \mathcal{L}(x_{1,j'}, \ldots, x_{n,j'})$ for $j' \neq j$, and $\tilde{x}'_j = \mathsf{HE.Eval}(\mathsf{pk}, \mathcal{L}, (\tilde{x}_{1,j}, \ldots, \tilde{x}_{n,j}))$, which constitutes essentially the j-th share of the value $\mathcal{L}(x_1, \ldots, x_n)$.

With the above observation in mind, we notice that if f is given in the $(\sum\prod\sum)$-form $f = \sum_{w=1}^{v} a_w \prod_{i=1}^{d} \mathcal{L}_{w,i}(X_1, \ldots, X_n)$, i.e., the sum of products of d linear functions over $X_1, \ldots, X_n$, the servers can first locally evaluate the linear functions and treat the result as shares of additional inputs, then apply the Split_d function on these new inputs. Note that even if f consists of only one product of d linear functions, the fully expanded form of f would contain $(n+1)^d$ monomials in general. This class of functions may be of particular interest since there exists a generic efficient transformation [23], due to an observation by Ben-Or, from any multilinear symmetric polynomial to depth-3 $(\sum\prod\sum)$-arithmetic circuits.

4.3 Efficiency Analysis

We analyze the efficiency of the construction in terms of server communication and computation complexity. The client-server communication is that of one ciphertext and therefore is independent from the size of the function that is computed, under the assumption that the underlying encryption scheme is *compact*. The input and output clients computation complexity is dominated by m calls to the encryption algorithm and one decryption, respectively.

The complexity of server computation depends on the design of the Split_d functions. Below, we first analyze a simpler greedy design where the workload is distributed to the servers unevenly. Next, we analyze a fair design in which the workload of each server is

identical. Surprisingly, the fair design seems to be worse than the greedy design in terms of computation complexity for $k > 1$.

We assume the polynomial f to be evaluated is given in the form $f = \sum_w a_w M_w(X_1, \ldots, X_n)$, where M_w are monomials of degree at most d. The efficiency analysis can be adapted easily to the setting where f is given as a sum of product of linear functions.

To bound the computation complexity of the servers, it is useful to use the following upper bounds, which can be verified straightforwardly:

$$\binom{n}{r_1, \ldots, r_m, n - r_1 - \ldots - r_m} \leq \frac{(en)^{r_1 + \ldots + r_m}}{r_1^{r_1} \cdot \ldots \cdot r_m^{r_m}} \tag{1}$$

$$\left(\frac{\alpha}{r}\right)^r \leq e^{\frac{\alpha}{e}} \tag{2}$$

where the multinomial coefficient $\binom{n}{r_1, \ldots, r_m, n - r_1 - \ldots - r_m}$ denotes the number of ways to distribute n distinct objects into $m + 1$ bins, with r_j objects in the j-th bin for $j \in [m]$, and $n - r_1 - \ldots - r_m$ objects in the last bin, and e is Euler's constant.

The Greedy Approach. One natural choice of Split_d is the greedy one: For each monomial $M_w = \prod_{i \in \{w_1, \ldots, w_c\}} \left(\sum_{j=1}^m x_{i,j}\right)$ in f, the first server (according to some fixed order) computes as many monomials in M_w as possible, then the second server computes as many monomials as possible except those which are already computed by the first server, and so on. We assume that each polynomial is given in the expanded form (as a sum of monomials). In turn, each monomial M_w is the product of several terms $X_{w_1} \cdots X_{w_c}$ for some $c \in [d]$, where each term X_i is shared as $X_i := x_{i,1} + \ldots + x_{i,m}$ for $i \in [n]$. This defines a circuit of depth 3 with $(n + 1)$ sum gates.

As computing sums are essentially free, we analyze the computation complexity of each of the servers, in terms of the number of product gates of the arithmetic circuit evaluated. It is useful to consider the following matrix:

$$T := \begin{bmatrix} x_{w_1,1} & x_{w_1,2} & \ldots & x_{w_1,m} \\ x_{w_2,1} & x_{w_2,2} & \ldots & x_{w_2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{w_c,1} & x_{w_c,2} & \ldots & x_{w_c,m} \end{bmatrix}.$$

All monomials in M_w can be obtained by multiplying c elements, such that each of which is chosen from a distinct row of T. The monomials that the first server can compute (homomorphically) consists of those obtained by multiplying at most k elements from the first column. One (efficient) way to compute the sum of these monomials is to first sum up the last $m - 1$ elements in each row i as $v_i = \sum_{j>1} x_{w_i,j}$, compute the products obtained by multiplying ℓ terms in $\{x_{w_1,1}, \ldots, x_{w_c,1}\}$ and $(c - \ell)$ terms in $\{v_1, \ldots, v_c\}$ for $\ell \in \{0, \ldots, k\}$, and sum up all products. The circuit for computing the above consists of $\sum_{\ell=0}^k \binom{c}{\ell}$ product gates.

The second server computes all monomials obtained by multiplying at most k elements from the second column, except those already computed by the first server. A way to compute this is first sum up the last $m - 2$ elements in each row i as $v_i = \sum_{j>2} x_{w_i,j}$,

compute the products obtained by multiplying ℓ_1 terms in $\{x_{w_1,1}, \ldots, x_{w_c,1}\}$, ℓ_2 terms in $\{x_{w_1,2}, \ldots, x_{w_c,2}\}$, and $\ell_3 = (c-\ell_1-\ell_2)$ terms in $\{v_1, \ldots, v_c\}$ for $\ell_1 \in \{k+1, \ldots, c\}$ and $\ell_2 \in \{0, \ldots, k\}$ such that $\ell_1 + \ell_2 \leq c$, and sum up all products. The number of product gates in the circuit for computing the above is given by

$$\sum_{\substack{\ell_1, \ell_2 : \\ \ell_1 + \ell_2 \leq c \\ \ell_1 \in \{k+1, \ldots, c\} \\ \ell_2 \in \{0, \ldots, k\}}} \binom{c}{\ell_1, \ell_2, c - \ell_1 - \ell_2}.$$

Proceeding this way, we can derive that the number of product gates in the circuit evaluated by the j-th server is given by

$$\begin{aligned}
&\sum_{\substack{\ell_1, \ldots, \ell_j : \\ \ell_1 + \ldots + \ell_j \leq c \\ \ell_1, \ldots, \ell_{j-1} \in \{k+1, \ldots, c\} \\ \ell_j \in \{0, \ldots, k\}}} \binom{c}{\ell_1, \ldots, \ell_j, c - \ell_1 - \ldots - \ell_j} \\
\leq &\sum_{\substack{\ell_1, \ldots, \ell_j : \\ \ell_1 + \ldots + \ell_j \leq c \\ \ell_1, \ldots, \ell_{j-1} \in \{k+1, \ldots, c\} \\ \ell_j \in \{0, \ldots, k\}}} \frac{c^{\ell_1 + \ldots + \ell_j}}{\ell_1^{\ell_1} \ldots \ell_j^{\ell_j}} \\
= &\sum_{\substack{\ell_1, \ldots, \ell_j : \\ \ell_1 + \ldots + \ell_j \leq c \\ \ell_1, \ldots, \ell_{j-1} \in \{k+1, \ldots, c\} \\ \ell_j \in \{0, \ldots, k\}}} \prod_{\ell \in \{\ell_1, \ldots, \ell_j\}} \left(\frac{c}{\ell}\right)^{\ell} \\
\leq &(k+1)(c-k-1)^{j-1} j e^{c/e} \\
\leq &(k+1)(c-k-1)^{m-1} m e^{c/e} \\
= &O(m^m) \\
= &O(2^{m \log m})
\end{aligned}$$

In order for the computation complexity of the servers to be polynomial, we set $m = O\left(\frac{\log \lambda}{\log \log \lambda}\right)$. Then, assuming f contains polynomially many monomials (or products of linear functions), the computation complexity of each server is bounded by

$$\mathsf{poly}(\lambda) \cdot O(2^{m \log m}) = O\left(2^{\frac{\log \lambda}{\log \log \lambda} \log\left(\frac{\log \lambda}{\log \log \lambda}\right)}\right) < O(2^{\log \lambda}) = \mathsf{poly}(\lambda)$$

The Fair Approach. Observe that in the greedy approach (the upper bound of) the workload of the j-th server increases as j increases, meaning that the distribution of work is unfair. This is undesirable since the overall computation time is determined by that of the slowest server. If the workload is distributed evenly, it might be possible that the workload of each server is lower than that of the slowest server in the greedy approach.

We denote by a vector $(\ell_1, \ldots, \ell_m)$ (where $\sum_{j \in [m]} \ell_j = d$) the classification of monomials obtainable by multiplying ℓ_j terms in the j-th column of T. For example,

consider the case $k = 1$, $m = 3$, and $d = (k+1)m - 1 = 5$. The monomial with classification $(1, 2, 2)$ can only be computed by the server with the first column encrypted, which is $\mathcal{S}_1$. Similarly, both $\mathcal{S}_1$ and $\mathcal{S}_2$ can compute monomials in the class $(0, 1, 4)$.

With the above observation in mind, we can design the Split_d function such that each server computes a weighted-sum of all monomials it can compute, where the monomials of a class that δ-many servers can compute are assigned the weight $1/\delta$ (assuming the message space R is also a field)[3]. The servers can thus group monomials of the same weight together, and try to reduce the number of multiplications as much as possible.

In the following, we describe one of the ways to group monomials, which is identical for all servers. Consider $\mathcal{S}_j$. To obtain the sum of all monomials which are only computable by $\mathcal{S}_j$, it chooses from each of the $m-1$ columns besides its own (column j) $k+1$ terms. This makes sure that no matter how the remaining k terms are chosen, the resulting monomials are only computable by $\mathcal{S}_j$. Due to the latter, it simply sum each of the remaining k rows where terms are not yet chosen, and multiply them to all $(k+1)(m-1)$ terms chosen in the beginning. Note that the number of ways to choose those $(k+1)(m-1)$ terms is given by

$$\begin{aligned} S_1 &:= \binom{m-1}{m-1}\binom{c}{\underbrace{k+1,\ldots,k+1}_{m-1},k} \\ &\leq \sum_{\ell_1=0}^{k}\binom{m-1}{m-1}\binom{c}{\underbrace{k+1,\ldots,k+1}_{m-1},\ell_1,c-(k+1)(m-1)-\ell_1}, \end{aligned}$$

where inequality will be useful for the analysis later. Summing all S_1 polynomials obtained by the above procedures (and assigning weight 1 to them) covers all monomials that are only computable by $\mathcal{S}_j$.

Moving on, to obtain the sum of all monomials which are only computable by $\mathcal{S}_j$ and one other servers, $\mathcal{S}_j$ chooses $m-2$ columns out of the other $m-1$ columns, and chooses $k+1$ terms each from these $m-2$ columns. Let j' be the column which is not chosen. It then chooses ℓ_1 items from its own column (from the remaining rows), and ℓ_2 items from column j', such that $\ell_1, \ell_2 \in \{0, \ldots, k\}$. This ensures that both $\mathcal{S}_j$ and $\mathcal{S}_{j'}$ and no server else can compute these monomials. Next, it sums up the elements in each of the remaining rows in the $m-2$ chosen columns, and multiplies each sum with the $(k+1)(m-2)+\ell_1+\ell_2$ terms chosen before. Note that the number of ways to choose those $(k+1)(m-2)+\ell_1+\ell_2$ terms is given by

$$S_2 := \sum_{\ell_1,\ell_2=0}^{k}\binom{m-1}{m-2}\binom{c}{\underbrace{k+1,\ldots,k+1}_{m-2},\ell_1,\ell_2,c-(k+1)(m-2)-\ell_1-\ell_2}.$$

Summing all S_2 polynomials obtained by the above procedures and assigning weight $1/2$ to them covers all monomials that are only computable by $\mathcal{S}_j$ and exactly one other server. Continue in this way, we conclude that the number of product gates for $\mathcal{S}_j$ is given by

[3] In general, it suffices for the servers to assign weights which add up to 1.

$$S := \sum_{i=1}^{m-1} \sum_{\ell_1,\ldots,\ell_i=0}^{k} \binom{m-1}{m-i} \binom{c}{\underbrace{k+1,\ldots,k+1}_{m-i}, \ell_1, \ldots, \ell_i, c-(k+1)(m-i)-\ell_1-\ldots-\ell_i}.$$

Using the inequality above, we have

$$\begin{aligned}
S &\leq \sum_{i=1}^{m-1} \sum_{\ell_1,\ldots,\ell_i=0}^{k} \left(\frac{e(m-1)}{(m-i)}\right)^{m-i} \frac{(ec)^{(k+1)(m-i)+\ell_1+\ldots+\ell_i}}{(k+1)^{(k+1)(m-i)}\ell_1^{\ell_1}\cdot\ldots\cdot\ell_i^{\ell_i}} \\
&= \sum_{i=1}^{m-1} \left(\frac{e(m-1)}{(m-i)}\right)^{m-i} \frac{(ec)^{(k+1)(m-i)}}{(k+1)^{(k+1)(m-i)}} \sum_{\ell_1,\ldots,\ell_i=0}^{k} \prod_{\ell\in\{\ell_1,\ldots,\ell_i\}} \left(\frac{ec}{\ell}\right)^{\ell} \\
&= \sum_{i=1}^{m-1} \left(\frac{e(m-1)(ec)^{(k+1)}}{(m-i)(k+1)^{(k+1)}}\right)^{m-i} (k+1)i \left(\frac{ec}{\ell}\right)^{\ell} \\
&\leq \frac{(k+1)m(m-1)}{2} \exp\left(\frac{(m-1)(ec)^{(k+1)}}{(k+1)^{(k+1)}} + c\right) \\
&\leq O(2^{m^k}).
\end{aligned}$$

In order for the computation complexity of the servers to be polynomial, we set $m = O\left(\log^{1/k}\lambda\right)$. Then, assuming f contains polynomially many monomials (or products of linear functions), the computation complexity of each server is bounded by

$$\mathsf{poly}(\lambda) \cdot O(2^{m^k}) = O(2^{(\log^{1/k}\lambda)^k}) = O(2^{\log\lambda}) = \mathsf{poly}(\lambda).$$

Note that for the case $k = 1$, *i.e.*, a linearly homomorphic encryption is used, we can set $m = O(\log\lambda)$, which is better than $m = O\left(\frac{\log\lambda}{\log\log\lambda}\right)$ set in the greedy approach.

4.4 Security Proof

We show that our construction is secure as per Definition 6 assuming HE is IND-CPA-secure. Furthermore, if HE is circuit-private, then our construction is context hiding.

Theorem 2. *Let* HE *be an IND-CPA-secure public key encryption scheme, then* HSS *constructed in Figure 2 is a secure* $(*, m, 1)$*-HSS scheme in the public-key setup model.*

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ which breaks the security of HSS with non-negligible probability, we show how to construct another efficient adversary $\mathcal{C}$ against the IND-CPA-security of HE.

$\mathcal{C}$ participates in the IND-CPA experiment of HE and receives pk which is forwarded to $\mathcal{A}$. The latter chooses $x_0^*, x_1^* \in R$, and an index $j^* \in [m]$. $\mathcal{C}$ samples $x_1, \ldots, x_{j^*-1}, x'_{j^*+1}, \ldots, x_m \leftarrow R$, and sets $x_{b,j^*} := x_b^* - \sum_{j\in[m]\setminus\{j^*\}}$ for $b \in \{0,1\}$. It then queries the challenge oracle of HE on (x_{0,j^*}, x_{1,j^*}), and receives in return $\tilde{x}_{j^*}$. Finally, it sends $s_{j^*} := (x_1, \ldots, x_{j^*-1}, \tilde{x}_{j^*}, x_{j^*+1}, \ldots, x_n)$ to $\mathcal{A}$. Eventually, $\mathcal{A}$ returns a bit b', which is forwarded by $\mathcal{C}$ to the IND-CPA experiment.

We analyze the success probability of $\mathcal{C}$ in breaking the IND-CPA-security of HE. By construction, if b is the bit chosen by the challenge oracle of HE, $\mathcal{C}$ simulates the $\mathsf{Security}^{b}_{\mathcal{A},\mathsf{HSS}}$ experiment for $\mathcal{A}$ faithfully, *i.e.*, the view of $\mathcal{A}$ simulated by $\mathcal{C}$ is identical to that in $\mathsf{Security}^{b}_{\mathcal{A},\mathsf{HSS}}$. Therefore, the probability of $\mathcal{C}$ guessing b correctly is identical to that of $\mathcal{A}$ breaking the security of HSS. This concludes our proof. □

Theorem 3. *Let* HE *be a circuit-private public key homomorphic encryption scheme, then* HSS *constructed in Figure 2 is a context-hiding* $(*, m, 1)$*-HSS scheme in the public-key setup model.*

Proof. We first describe the simulator $\mathcal{S}$: On input the security parameter 1^λ, the public key pk, and the function output r, the simulator $\mathcal{S}$ samples some random $(r_1, \ldots, r_m)$ under the constraint that $\sum_{j\in[m]} r_j = r$ and executes $c_j \leftarrow \mathcal{S}_{\mathsf{HE}}(1^\lambda, \mathsf{pk}, r_j)$, for all $j \in [m]$, where $\mathcal{S}_{\mathsf{HE}}$ is the simulator of HE. The simulator $\mathcal{S}$ returns $(c_1, \ldots, c_m)$.

We analyze the distribution of the output of the simulator $\mathcal{S}$. Consider the output of the Eval algorithm, for all $j \in [m]$. By the circuit privacy of HE we have that:

$$\begin{aligned} y_j &= \mathsf{HE.Eval}(\mathsf{pk}, f_j, (\tilde{x}_{1,j}, \ldots, \tilde{x}_{n,j})) \\ &\approx \mathcal{S}_{\mathsf{HE}}\left(1^\lambda, \mathsf{pk}, f_j(x_{1,j}, \ldots, x_{n,j})\right) \\ &= \mathcal{S}_{\mathsf{HE}}\left(1^\lambda, \mathsf{pk}, g_j(x_{1,j}, \ldots, x_{n,j}) + \sum_{i\in[n]} z_{i,j}\right) \end{aligned}$$

where $\approx$ denotes statistical indistinguishability. Consider any subset $M \subseteq [m]$ of size $m-1$. Then for all $j \in M$ we have that:

$$\begin{aligned} y_j &\approx \mathcal{S}_{\mathsf{HE}}\left(1^\lambda, \mathsf{pk}, g_j(x_{1,j}, \ldots, x_{n,j}) + \sum_{i\in[n]} z_{i,j}\right) \\ &\approx \mathcal{S}_{\mathsf{HE}}\left(1^\lambda, \mathsf{pk}, r_j\right) \end{aligned}$$

for some $r_j \in R$ sampled uniformly at random, since there exists at least one (in fact all) $i \in [n]$ such that $z_{i,j}$ is sampled uniformly and independently in R. By the correctness of HSS it must be the case that for $j \notin M$:

$$y_j \approx \mathcal{S}_{\mathsf{HE}}\left(1^\lambda, \mathsf{pk}, \sum_{j\in[m]} r_j - r\right)$$

which is exactly the output of the simulator $\mathcal{S}$. □

Multi-Use Context-Hiding. We point out that the standard definition of context-hiding takes into account only one execution of the Eval algorithms, whereas in certain scenarios it might be desirable to preserve context-hiding even when multiple functions are evaluated over the same shares. We propose a simple modification of our scheme that achieves the stronger version of the property: Instead of computing the shares for the

value 0, the sharing algorithm initializes m keys for a certain pseudo-random function PRF $(\kappa_{i,1}, \ldots, \kappa_{i,m})$ and each server j is given $(\kappa_{i,j}, \kappa_{i,(j+1 \mod m)})$. Then, for all $j \in [m]$, the function f_j is defined as

$$f_j := \mathsf{Split}_d(j, f, (x_1^{-j}, \ldots, x_n^{-j})) + \sum_{i \in [n]} \mathsf{PRF}(\kappa_{i,j}, f) - \sum_{i \in [n]} \mathsf{PRF}(\kappa_{i,(j+1 \mod m)}, f).$$

The analysis follows, with minor modifications, along the lines of what discussed above.

5 Multi-Key Construction in the Plain Model

In the following we show how to extend the scheme in Section 4 to the multi-key settings (in the public-key setup model) and how to turn it into a plain model scheme.

5.1 Intuition

First, we observe that the main construction in Section 4 can be naturally extended to the multi-key setting, where shares under different public keys are combined in the evaluation algorithm. In this context, it is useful to distinguish between *input* and *output* clients: The former provide the input data and share them to the same set of servers[4] whereas the latter decode the output of the computation. Note that these two sets of clients may intersect arbitrarily. We stress that the clients are not assumed to communicate with each other and can generate their input shares independently. Adapting our protocol to this setting is surprisingly simple: In a nutshell, it is sufficient to replace the homomorphic encryption scheme with the corresponding multi-key variant.

Next, we turn the multi-key construction into a plain model construction. The idea is to let the *input clients*, instead of the output clients, generate in the share algorithm a fresh pair of public and secret keys. They then secret share their data under the freshly generated public key as in the multi-key construction, and further secret share the fresh secret keys to all m servers using an m-out-of-m secret sharing scheme. The servers evaluate the shares as in the multi-key construction, and forward the output shares along with the shares of the secret keys to the output client. The latter recovers the secret keys and uses them to decode the output shares as in the multi-key construction.

5.2 Construction

Below, we describe briefly the modifications made to the construction in Section 4 to obtain a plain model scheme. A formal description is given in Figure 3. Let m and k be positive integers. We present a generic construction of an unbounded-input (1-output) m-server 1-secure homomorphic secret sharing ($(*, m, 1)$-HSS) scheme pHSS for degree-d polynomials $\mathcal{R}_d$ in the plain model, where $d = (k+1)m - 1$, using only a public-key multi-key homomorphic encryption scheme HE for degree-k polynomials.

Key Generation. In the plain model, key generation is no longer needed.

[4] While sharing to different sets of servers is in general possible, it limits the class of polynomials that can be computed. Specifically, if there exists a server picked by client i but not client j, then any polynomial which contains a product of data contributed by both clients is not computable.

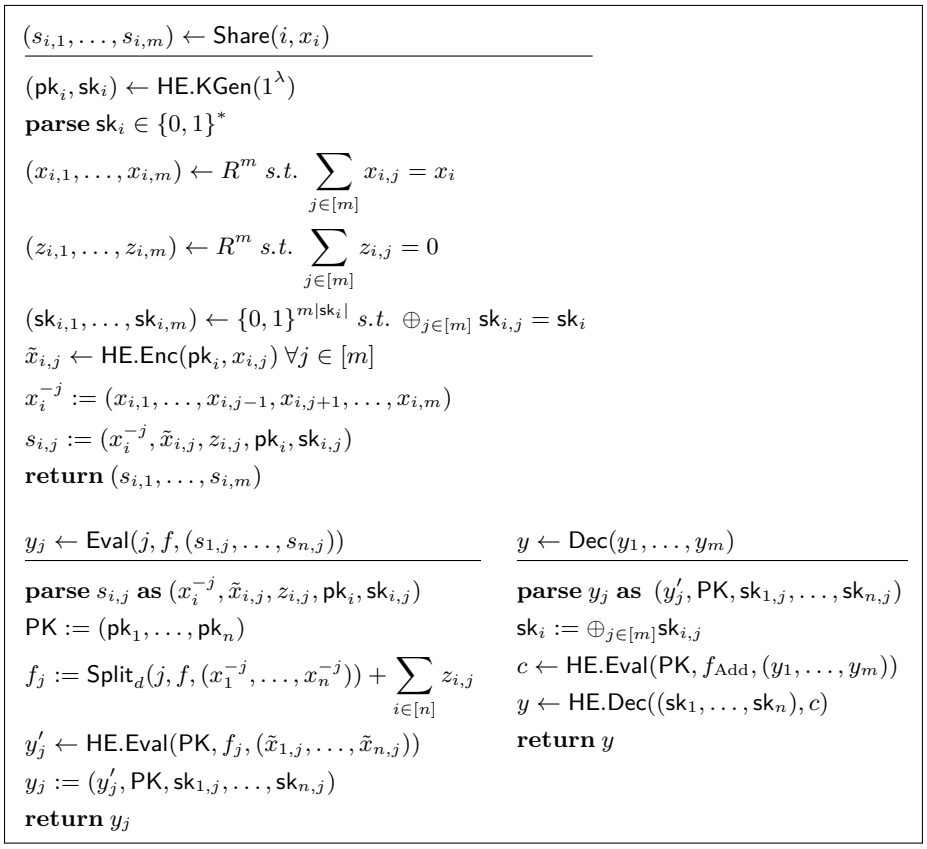

Fig. 3: Construction of a homomorphic secret sharing scheme pHSS in the plain model. (The functions f_{Add} and Split_d are defined in Section 4.1.)

Secret Sharing. An input client runs the key generation algorithm for a multi-key homomorphic encryption scheme to generate a public key and a secret key independent of other input clients. It then runs the same sharing algorithm (under the generated public key) to share its private data. It also secret-shares the secret key of the encryption scheme using an m-out-of-m secret sharing scheme. Finally, it appends the public key and the j-th share of the secret key to the j-th input share given to the j-th server.

Evaluation. The evaluation performed by the servers is almost identical, except that the evaluation algorithm of the multi-key homomorphic encryption scheme inputs ciphertexts encrypted under different pubic keys and outputs a ciphertext encrypted under the set of combined public keys. The shares of the secret keys remain untouched, and are forwarded to the output client along with the output of the homomorphic evaluation.

Decoding. The output client collects all shares of all secret keys, and recovers them. As in the previous construction, it homomorphically evaluates the output shares received

from the servers and obtain a ciphertext encrypting the computation result. The only difference is that now the result is encrypted under a set of public keys. The output client thus uses all the recovered secret keys to decrypt the ciphertext and obtain the result.

The analyses of correctness, efficiency, and security are almost identical to those in Section 4. We thus state the formal results and omit the proofs.

Theorem 4. *Let* HE *be an IND-CPA-secure public-key encryption scheme, then* pHSS *constructed in Figure 3 is a secure* $(*, m, 1)$*-HSS scheme in the plain model.*

Theorem 5. *Let* HE *be a circuit-private public-key multi-key homomorphic encryption scheme, then* pHSS *constructed in Figure 3 is a context-hiding* $(*, m, 1)$*-HSS scheme in the plain model.*

6 Collusion-Resistance

The constructions in Section 4 and Section 5 are 1-secure, meaning that security is lost as soon as two servers collude. We outline how the construction can be upgraded to give a $(*, m, t)$-HSS scheme which tolerates $t > 1$ colluding servers, and investigate the effect on the supported degree d of the resulting secret sharing scheme.

Bounding the Number of Plaintext Base Shares. Unlike in the previous constructions, where we use an m-out-of-m secret sharing scheme to generate base shares of each input x_i, we now use a b-out-of-b secret sharing scheme instead, where b is a new independent variable. Suppose that for each x_i, a certain choice of p out of b base shares of x_i are given in plaintext to a server. In the previous constructions, $p = b - 1$. This means any two colluding servers collectively possess all base shares of x_i in plaintext, and hence are able to recover x_i. To tolerate t colluding servers, we must set p and b such that $b > tp$, so that any t colluding servers collectively possess at most $tp < b$ out of b base shares of each x_i in plaintext, and are thus unable to recover any x_i.

Bounding the Supported Degree of the Homomorphic Secret Sharing Scheme. Next, we analyze the supported degree d of the resulting HSS scheme, assuming an encryption scheme supporting degree k is used. Recall the matrix representation of the shares as defined in Section 4.3. The goal of the servers is to jointly compute (homomorphically) the product of the sums of each row, which can be rewritten as a sum of the products obtained by choosing one element from each row. A product of d-many rows (and therefore a degree d polynomial) is computable only if, for each monomial of such a product, there exists at least one server where at most k elements of such a monomial belong to the encrypted columns possessed by this server. A natural strategy to maximize the degree is to let the server possessing the highest number of plaintext columns to compute such a monomial. Let us rewrite $k = (b - p)u + v$, for some quotient u and remainder v. A "worst case" configuration is visualized as follows.

$$\overbrace{\underbrace{u, \ldots, u}_{b-p-v}, \underbrace{u+1, \ldots, u+1}_{v}}^{\text{Encrypted Columns, } k \text{ elements}}, \quad \overbrace{\underbrace{u+1, \ldots, u+1}_{p}}^{\text{Plaintext Columns, } d-k \text{ elements}}$$

The above means that u elements are chosen from each of the $b-p-v$ of the encrypted columns, and $u+1$ elements are chosen from each of the remaining plaintext and encrypted columns. Consider shifting any element from a plaintext column to an encrypted column j^-. We argue that (after the shift) there exists a configuration of $b-p$ encrypted columns with at most k elements in the encrypted columns. Such a configuration is obtained as follows: Let j^+ be the plaintext column with the least amount of elements after the shift. We move j^- to the set of plaintext columns and j^+ to the set of encrypted columns. The numbers of plaintext and encrypted columns clearly do not change. Since j^- has at least $u+1$ elements and j^+ has at most $u+1$ elements, there is no positive gain in elements in the set of encrypted columns. Thus the new configuration has at most k elements in the encrypted columns, as the previous configuration does. This shows that the case constructed above is indeed the worst case. Assume for the moment that each monomial is computable by at least one server, then the supported degree cannot exceed

$$\begin{aligned} d =& \ bu + p + v \\ =& \ b\left(\frac{k-v}{b-p}\right) + p + v \\ =& \ \frac{bk}{b-p} + v\left(1 - \frac{b}{b-p}\right) + p \\ =& \ \frac{bk}{b-p} - v\left(\frac{p}{b-p}\right) + p. \end{aligned}$$

Depending on the value of v, which is uniquely determined by (k, b, p) and satisfies $0 \leq v < b-p$, the maximum supported degree lies within the range

$$k \cdot \frac{b}{b-p} < d \leq k \cdot \frac{b}{b-p} + p.$$

To maximize the above range, we can fix $p = \frac{b-1}{t}$, and have

$$k \cdot \frac{tb}{(t-1)b+1} < d \leq k \cdot \frac{tb}{(t-1)b+1} + \frac{b-1}{t}. \tag{3}$$

For consistency check, we can differentiate with respect to t. For the (non-trivial) case where $p \geq 1$, we have $1 \leq t \leq b-1$, and hence

$$k < \frac{kb}{b-1} < d \leq (k+1)b - 1.$$

When no collusion is allowed, *i.e.*, $t = 1$, we can set $b = m$ and recover the previous bound $d \leq (k+1)m-1$. This bounds the maximum supported degree by $(k+1)m-1$, regardless of how many servers are involved. The constructions in Section 4 and Section 5, show that this bound is actually achievable using m servers.

If a collusion of two servers is allowed, *i.e.*, $t \geq 2$, we examine the bound given in Equation 3. An interpretation is that the construction amplifies the supported degree of the base encryption scheme by roughly $t/(t-1)$ multiplicatively (by taking limit as the number of columns $b \to \infty$), then adds roughly $1/t$ degree per column.

Bounding the Minimum Number of Servers. It remains to show the condition for having each monomial computable by at least one server. Fix b, p, k and d. There must exists an integer δ such that $d \leq \delta \cdot k$. We argue that the number of servers m required is lower bounded by the solution of the following set cover problem.

From the set of all $\binom{b}{p}$ number of configurations of choosing p plaintext columns out of b columns in a secret share matrix, choose a subset satisfying the following properties: For any integer $s \in [b]$, any combination of s out of all b columns, any combination of $\lfloor s/\delta \rfloor$ out of these s columns, there exists at least one configuration (a secret share matrix) in the subset which has at most these $\lfloor s/\delta \rfloor$ columns (out of the s columns) encrypted. Each satisfying subset specifies a set of servers.

We argue that the above condition on m is both necessary and sufficient, *i.e.*, the lower bound is tight. For the former, suppose that the condition is not satisfied, namely that there exists s, a combination of s columns, and a combination of $\lfloor s/\delta \rfloor$ out of these s columns, such that the secret share matrices of all servers have more than these $\lfloor s/\delta \rfloor$ columns encrypted. Consequently, no server is able to compute the monomial where all elements are contributed from these s/δ columns.

For the sufficiency, consider without loss of generality any degree d monomial, and denote the number of columns contributing elements to this monomial by $s \in [b]$. Since $d \leq \delta \cdot k$, there must exist $\lfloor s/\delta \rfloor$ columns out of these s columns that are contributing at most k elements. By the condition specified above, there must exist a server whose secret share matrix has at most these $\lfloor s/\delta \rfloor$ columns encrypted (whereas the other $s - \lfloor s/\delta \rfloor$ are in plaintext). This server is thus able to compute the monomial.

Although the general set cover problem is NP-hard, the greedy algorithm is known to solve the general problem with (multiplicative) approximate factor $O(\log N)$ [16] , where N is the number of elements to be covered.

Practically-Relevant Parameters. The above analysis does not give a close form for the number of servers m needed, for a fixed set of parameters (t, b, p, k, d). In the following, we investigate parameter settings which are most practically relevant. First, we restrict ourselves to use only encryption for affine functions, *i.e.*, $k = 1$. Next we consider the computation of polynomials of degree $d = 3$, since they are sufficient for the secure computation of any function via randomized encodings [1]. In order to compute degree-3 polynomials, we must have $p > 1$[5]. We pick $p = 2$ and set $b = 2t+1$, where $t > 1$. Using a greedy algorithm, it can be found that the number of servers needed for $t = 2, 3, 4$ are $m = 4, 9, 16$ respectively, which seems to suggest that $m \approx t^2$. If that is the case, the maximum tolerated ratio of colluding servers is $t/m \approx 1/t$, which gets worse as t grows. Therefore, a $(*, 4, 2)$-HSS for degree-3 polynomials from affine encryption seems to be the most interesting result in terms of collusion resistance.

7 Applications

We highlight several interesting applications of our HSS scheme.

[5] If $p = 1$, then the best the Split_d function can do is to assign one element to the plaintext column and one element to one of the ciphertext columns, whose product is of degree 2.

7.1 Server-aided Secure Evaluation of Low-Degree Polynomials

Server-aided secure computation is a natural application of HSS schemes. In this scenario, one or multiple input clients secret share their data to a set of servers. Later, the servers can homomorphically evaluate functions (*e.g.*, given by the input clients or other parties) on the shared data and send the result to an output client. The latter can efficiently recover the computation result, and due to the context-hiding property, without learning the original data and the function being evaluated beyond what is trivially revealed by the result. Using an HSS scheme in this scenario is particularly appealing since the client-server communication is succinct (independent of the size of the function evaluated) and the workload of the input and output clients is typically small.

When instantiated with linearly homomorphic encryption schemes such as ElGamal [19] or Paillier [29], our main construction allows input clients to outsource the computation of degree-d polynomials to m non-communicating servers, where $d = 2m - 1$ and $m = O(\log \lambda)$. Since Shamir secret sharing [30] allows to evaluate polynomials of degree $d = m - 1$ using $m = \mathsf{poly}(\lambda)$ servers, our result is more interesting when the number of servers, and hence the degree of the polynomials, are small. There are a few interesting scenarios where low-degree polynomials are evaluated over private data.

1. *Moments:* Moments are recurrent measures in statistics and physics to describe the shape of a set of points. The d-th moment is computable by a degree-d polynomials. The mean is the first row moment and the variance and the skewness are the second and third central moments, respectively. Notably, our scheme allows two servers to efficiently compute the third moment from standard assumptions whereas previous approaches with comparable efficiency [15] rely on bilinear maps.
2. *Neural Networks:* Multi-layered non-recurrent neural networks are arithmetic circuits consisting of gates computing non-linear functions. Previous work on privacy-preserving neural network evaluation [24] approximates these non-linear functions using low-degree polynomials, such that the modified neural network can be homomorphically evaluated using a fully homomorphic encryption [22] (FHE) scheme with a reasonable parameter. Suppose that the networks to be evaluated are shallow enough, then our constructions provide a relatively lightweight alternative to FHE for evaluating neural networks over private data.
3. *Polynomials with Hidden Coefficients:* Suppose the polynomial f of degree d to be evaluated is given also by the input clients, they can choose to hide the coefficients of the monomials in f by secret sharing them, and turning f into a new polynomial f' of degree $d + 1$, where the coefficients in f become variables in f'. The clients can further hide the monomials appearing in f by secret sharing the (possibly zero) coefficients in f of all monomials of degree at most d. Note that although the number of such monomials is exponential in d, it is not an issue for a degree $d = O(\log \lambda)$ (when $k = 1$ and $m = O(\log \lambda)$) which is logarithmic in the security parameter.
4. *m-Server PIR:* Our HSS scheme for degree d polynomials can be easily converted into a (round-optimal) m-server PIR scheme as follows: Consider a set of m servers who store a copy of a database DB locally and let us split the database in 2^d equal chunks $(\mathsf{DB}_1, \ldots, \mathsf{DB}_{2^d})$, then, on input an index $i \in \{0,1\}^d$, the client shares i to

the m servers with our HSS scheme. The servers evaluate the function

$$g(i) := \sum_{j=1}^{2^d} \mathsf{DB}_j(i = j) = \mathsf{DB}_i$$

and send the output of the computation to the client. Note that g is a polynomial of degree d with coefficients determined by DB, which is public. The communication complexity is dominated by the factor $\frac{|\mathsf{DB}|}{2^d} + \mathsf{poly}(\lambda)$ of the server-client message.

7.2 Round-Optimal Server-Aided Multiparty Computation in the Plain Model

A recent result by Boyle *et al.* [11] shows that an additive $(3, 2, 1)$-HSS schemes for degree-3 polynomials and a low-depth PRG imply a 2 round (n, m)-MPC protocols[6], where $n, m \in \mathsf{poly}(\lambda)$. Since the only $(3, 2, 1)$-HSS scheme from standard assumption was known to exist only in the PKI model [9], the resulting MPC protocol inherits the same setup assumption. Unfortunately the transformation assumes a linear reconstruction of the HSS, which is not satisfied by our scheme.

However, we can apply a similar transformation to our multi-client HSS scheme in Section 5 to obtain a 2 round (n, m)-MPC, where the adversary is allowed to corrupt any strict subset of the servers *or* the output client, and an arbitrary number of input clients. This corruption model has been introduced in the context of server-aided multiparty-computation [28]. We denote such a primitive by (n, m)-saMPC. Our scheme does not require a PKI and can be instantiated in the plain model. Moreover, since our HSS scheme is perfectly correct (assuming a perfectly correct homomorphic encryption scheme), we can avoid the probability amplification step in [10]. We briefly outline the steps of the transformation in the following.

1. $(n, 2, 1)$-*HSS for degree 3 polynomials* $\implies$ $(n, 2, 1)$-*HSS for P/poly.* This is a trivial implication using randomized encodings [1] and assuming the existence of a low-depth PRG.
2. $(n, 2, 1)$-*HSS for P/poly* $\implies$ $(n, 3)$-*saMPC for P/poly.* This is shown using the server-emulation technique described in [10], where the inputs of one server are secret shared among two new servers and its computation is emulated using the $(n, 2, 1)$-HSS. Note that the resulting $(n, 3)$-saMPC is resilient against the corruption of any strict subset of the 3 servers *or* the output client.
3. $(n, 3)$-*saMPC for P/poly* $\implies$ (n, m)-*saMPC for degree 3 polynomials.* This is shown using the following observation of [11]: Given a degree 3 polynomial $f(X_1, \ldots, X_n)$, then rewriting $X_j = \sum_{i=1}^{m} x_{j,i}$ we obtain another degree 3 polynomial $f(\sum_{i=1}^{m} x_{1,i}, \ldots, \sum_{i=1}^{m} x_{n,i})$. Each monomial is of the form $x_{1,i}x_{2,j}x_{2,k}$ and can be computed by the servers $(\mathcal{S}_i, \mathcal{S}_j, \mathcal{S}_k)$ with the $(n, 3)$-saMPC scheme. Padding each monomial with a blinding factor (such that all factors sum up to 0) gives us the final (n, m)-saMPC protocol for degree 3 polynomials.
4. (n, m)-*saMPC for degree 3 polynomials* $\implies$ (n, m)-*saMPC for P/poly.* Follows by another application of randomized encodings.

[6] (n, m)-MPCs are n-client m-server MPCs which are secure against $m - 1$ corrupt server.

References

1. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *computational complexity*, 2006.
2. László Babai, Peter G Kimmel, and Satyanarayana V Lokam. Simultaneous messages vs. communication. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 361–372. Springer, 1995.
3. Omer Barkol, Yuval Ishai, and Enav Weinreb. On d-multiplicative secret sharing. *Journal of Cryptology*, 23(4):580–593, October 2010.
4. Amos Beimel and Yuval Ishai. Information-theoretic private information retrieval: A unified construction. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *ICALP 2001: 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 912–926. Springer, Heidelberg, July 2001.
5. Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Ilan Orlov. Share conversion and private information retrieval. In *Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012*, pages 258–268. IEEE Computer Society, 2012.
6. Josh Cohen Benaloh. Secret sharing homomorphisms: Keeping shares of a secret sharing. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 251–260. Springer, Heidelberg, August 1987.
7. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, Heidelberg, February 2005.
8. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 337–367. Springer, Heidelberg, April 2015.
9. Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 509–539. Springer, Heidelberg, August 2016.
10. 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, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 163–193. Springer, Heidelberg, May 2017.
11. Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In *Innovations in Theoretical Computer Science*, volume 94. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
12. Christian Cachin, Jan Camenisch, Joe Kilian, and Joy Müller. One-round secure computation and secure autonomous mobile agents. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *ICALP 2000: 27th International Colloquium on Automata, Languages and Programming*, volume 1853 of *Lecture Notes in Computer Science*, pages 512–523. Springer, Heidelberg, July 2000.
13. Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer, Heidelberg, May 1999.
14. Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In Kaisa Nyberg, editor, *Topics in Cryptology – CT-RSA 2015*, volume 9048 of *Lecture Notes in Computer Science*, pages 487–505. Springer, Heidelberg, April 2015.
15. Dario Catalano and Dario Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors,

ACM CCS 15: 22nd Conference on Computer and Communications Security, pages 1518–1529. ACM Press, October 2015.
16. V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.
17. Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 93–122. Springer, Heidelberg, August 2016.
18. Klim Efremenko. 3-query locally decodable codes of subexponential length. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 39–44. ACM Press, May / June 2009.
19. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, Heidelberg, August 1984.
20. Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith. Homomorphic secret sharing from paillier encryption. In *Provable Security*, 2017.
21. Matthew K. Franklin and Payman Mohassel. Efficient and secure evaluation of multivariate polynomials and applications. In Jianying Zhou and Moti Yung, editors, *ACNS 10: 8th International Conference on Applied Cryptography and Network Security*, volume 6123 of *Lecture Notes in Computer Science*, pages 236–254. Springer, Heidelberg, June 2010.
22. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178. ACM Press, May / June 2009.
23. Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 107–109. IEEE Computer Society Press, October 2011.
24. Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.
25. Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 640–658. Springer, Heidelberg, May 2014.
26. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.
27. Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011. http://eprint.iacr.org/2011/272.
28. Seny Kamara, Payman Mohassel, and Ben Riva. Salus: a system for server-aided secure function evaluation. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12: 19th Conference on Computer and Communications Security*, pages 797–808. ACM Press, October 2012.
29. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, Heidelberg, May 1999.
30. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
31. Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing*, pages 266–274. ACM Press, June 2007.