Homomorphic Secret Sharing from Lattices Without FHE

Elette Boyle[1] *, Lisa Kohl[2] †, and Peter Scholl[3] ‡

[1] IDC Herzliya
Elette.Boyle@idc.ac.il

[2] Karlsruhe Institute of Technology
Lisa.Kohl@kit.edu

[3] Aarhus University
Peter.Scholl@cs.au.dk

Abstract. Homomorphic secret sharing (HSS) is an analog of somewhat- or fully homomorphic encryption (S/FHE) to the setting of secret sharing, with applications including succinct secure computation, private manipulation of remote databases, and more. While HSS can be viewed as a relaxation of S/FHE, the only constructions from lattice-based assumptions to date build *atop* specific forms of threshold or multi-key S/FHE. In this work, we present new techniques directly yielding efficient 2-party HSS for polynomial-size branching programs from a range of lattice-based encryption schemes, *without S/FHE*. More concretely, we avoid the costly *key-switching* and *modulus-reduction* steps used in S/FHE ciphertext multiplication, replacing them with a new *distributed decryption* procedure for performing "restricted" multiplications of an input with a partial computation value. Doing so requires new methods for handling the blowup of "noise" in ciphertexts in a distributed setting, and leverages several properties of lattice-based encryption schemes together with new tricks in share conversion.

The resulting schemes support a superpolynomial-size plaintext space and negligible correctness error, with share sizes comparable to SHE ciphertexts, but cost of homomorphic multiplication roughly one order of magnitude faster. Over certain rings, our HSS can further support some level of packed SIMD homomorphic operations. We demonstrate the practical efficiency of our schemes within two application settings, where we compare favorably with current best approaches: 2-server private database pattern-match queries, and secure 2-party computation of low-degree polynomials.

*Supported in part by ISF grant 1861/16, AFOSR Award FA9550-17-1-0069, and ERC grant 742754 (project NTSC).

†Supported by ERC Project PREP-CRYPTO (724307), by DFG grant HO 4534/2-2 and by a DAAD scholarship. This work was done in part while visiting the FACT Center at IDC Herzliya, Israel.

‡Supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 731583 (SODA), and the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC).

1 Introduction

Homomorphic secret sharing (HSS) [7] is a form of secret sharing that supports a compact local evaluation on its shares. HSS can be viewed as the analog of fully (or somewhat-) homomorphic encryption (S/FHE) [38, 26] to the setting of secret sharing: a relaxation where homomorphic evaluation can be distributed among two parties who do not interact with each other. Over the past years, there has been a wave of HSS constructions for rich function classes (e.g., [7, 9, 25, 6, 21]) as well as an expanding range of corresponding applications. HSS suffices for many scenarios in which S/FHE can be applied (and even some for which it *cannot*), including low-communication secure computation [7, 10, 6], private manipulation of remote databases [29, 8, 19, 9, 40], methods of succinctly generating correlated randomness [6, 5], and more.

One of the appealing features of HSS compared to FHE is that allowing homomorphic evaluation to be distributed among two parties may constitute a simpler target to achieve. Indeed, forms of HSS for branching programs have been built from *discrete logarithm* type assumptions [7]; in contrast, obtaining encryption schemes from these structures that support comparable homomorphism on ciphertexts seems well beyond reach of current techniques. In regard to structures from which FHE does exist, the Learning With Errors (LWE) assumption [37] (and in turn its Ring LWE (RLWE) variant [36]) is known to imply strong versions of HSS [8, 22, 11].

However, in spite of its potential for comparative simplicity, all HSS constructions based on LWE or RLWE to date remain *at least* as complex as S/FHE. In particular, underlying each such HSS scheme is the common approach of beginning with and building atop some existing construction of FHE—relying on specific forms of threshold FHE, multi-key FHE, or even FHE-based "spooky encryption" [2, 8, 22, 11]. In this sense, current lattice-based HSS constructions serve predominantly as statements of feasibility, and have not been explored as a competitive alternative for use within applications.

Given the rapidly expanding set of HSS applications, together with the demonstrated power and success of leveraging lattices as a tool for advanced cryptography, a natural question is whether this situation can be improved. In particular, can we construct HSS from LWE and RLWE *without* (in some sense) S/FHE?

1.1 Our Results

In this work we consider precisely this question. We present and leverage new approaches for directly obtaining 2-party HSS schemes from LWE and RLWE, bypassing the intermediate step of fully (or even somewhat-) homomorphic encryption (S/FHE).

More concretely, our techniques avoid the costly *key-switching* and repeated *modulus-reduction* steps typically required for homomorphic multiplication of ciphertexts in existing (R)LWE-based FHE schemes [14, 27], and replace them instead with a new *distributed decryption* procedure for multiplying an encrypted

value by a value in secret shared form, resulting in secret shares of the product. The cost of a homomorphic multiplication thus drops roughly to the cost of a decryption operation per party. This operation requires a new toolkit of methods for handling the blowup of "noise" from ciphertexts in a distributed setting, and leverages properties of lattice-based encryption schemes (such as key-dependent message security) in new ways.

Our construction takes inspiration from the HSS framework of [7], and yields a similar result: namely, HSS for the class of polynomial-size branching programs (capturing $\mathsf{NC1}$ and logspace computations). However, as discussed below, our construction offers several strong advantages over existing DDH-based schemes [7, 10, 6, 21], including *negligible* correctness error and *superpolynomial*-size plaintext space, as well as over S/FHE-based solutions for the same program class [2, 22], including cheaper multiplication, simpler setup, and *no noise growth*. We showcase these advantages via two sample applications: (1) Generating correlated randomness for secure 2-party computation in the prepocessing model, and (2) 2-server Private Information Retrieval for various private database queries such as conjunctive keyword search and pattern matching.

We now proceed to describe our main results.

HSS from Nearly Linear Decryption. Our core approach leverages the *"nearly linear"* structure of ciphertext decryption common to a range of lattice-based encryption schemes:

Definition 1 (Informal - *Nearly Linear Decryption*). *Let $R = \mathbb{Z}[X]/(X^N + 1)$ for N a power of 2. Let $p, q \in \mathbb{N}$ be moduli with $p|q$ and $1 \ll p \ll q$. We say that an encryption scheme supports* nearly linear decryption *for messages $m \in R_p := R/pR$ if the secret key is $\mathbf{s} \in R_q^d$, and for any ciphertext $\mathbf{c} \in R_q^d$ encrypting m,*

$$\langle \mathbf{s}, \mathbf{c} \rangle = (q/p) \cdot m + e \mod q$$

for some "small" noise $e \in R$.

This captures, for example, the LWE-based schemes of Regev [37] and Applebaum *et al.* [1] (with $N = 1$, i.e. $R = \mathbb{Z}$, $d = \mathsf{poly}(\lambda)$), RLWE-based schemes of Lyubashevsky-Peikert-Regev [36] and Brakerski-Vaikutanathan [15] (with $N = \mathsf{poly}(\lambda), d = 2$), as well as various schemes based on Learning With Rounding [3] and Module-LWE [34]. For simplicity, we restrict ourselves to the decryption structure and polynomial rings R of the form specified above; however, our techniques extend to more general polynomial rings, as well as to encryption schemes which encode messages in low-order symbols (i.e., for which $\langle \mathbf{s}, \mathbf{c} \rangle = p \cdot e + m \mod q$).

We demonstrate how to exploit the near-linearity of decryption to support sequences of homomorphic *additions* over R, as well as homomorphic *"restricted" multiplications* over R of an evaluated value with an input. Ultimately, our scheme supports any sequence of these two homomorphic operations over R_r (for $r \in \mathbb{N}$ of choice) with negligible correctness error, subject to the requirement that the magnitude of computation values remain bounded by some chosen

value B with $r \leq B \ll p$ (where the required size of p, and thus q, must grow with this bound). We remark that taking magnitude bound $B = 2$ suffices already to evaluate the class of polynomial-size branching programs with polynomial overhead [7]. Since efficiency is the primary focus of this work, however, we state our results as a function of these two operations directly, and applications can further exploit the ability to support large message spaces.

We achieve the stronger "public-key" variant of HSS [7], where the secret-sharing process can be split into a one-time setup phase (resulting in a public key pk and evaluation shares $(\mathsf{ek}_0, \mathsf{ek}_1)$) together with a separate "input encryption" phase, wherein any user can use pk to load his input x_i into secret-shared form (and where homomorphic evaluation can take place *across* parties' inputs). This variant of HSS facilitates applications of secure multi-party computation.

Theorem 1 (Informal - *Main HSS Construction*). *Given any encryption scheme with nearly linear decryption (as above) over ring R with parameters $p, q, d \in \mathbb{N}$, as well as magnitude bound $B \in \mathbb{N}$ for which $B \ll p \ll q/B$, and output modulus $r \leq B$, there exists 2-party public-key HSS for inputs in R_r with size:*

- *Public key $\mathsf{pk} = pk$ of the encryption scheme, Evaluation keys $\mathsf{ek}_b \in R_q^{d-1}$*
- *HSS shares of each input $x_i \in R_r$ consist of d ciphertexts.*

supporting any polynomial number of the following homomorphic operations over R_r (subject to the ℓ_∞ magnitude bound $\|y\|_\infty \leq B$ (in R) for all partial computation values y), with negligible correctness error, and the specified complexities:

- *Loading an input into memory ($y_i \leftarrow x_i$):* d *decryptions*
- *Addition of memory values ($y_k \leftarrow y_i + y_j$):* 1 *addition over* R_q^d
- *Multiplication of input with memory value ($y_k \leftarrow x_i \cdot y_j$):* d *decryptions*
  "Terminal" multiplication (s.t. y_k appears in no future mult): 1 *decryption*

where "decryption" is essentially one inner product over R_q^d.

Asymptotically, the modulus q is of size $O(\lambda + \log B)$ bits, giving a share size in $O(Nd(\lambda + \log B))$ for $R = \mathbb{Z}[X]/(X^N + 1)$. The cost of multiplication is $\widetilde{O}(Nd)$ operations in $\mathbb{Z}_q$.

Plugging in the "LPR" RLWE-based scheme of [36] (where ciphertexts consist of $d = 2$ ring elements), our HSS shares consist of 4 R_q-elements per R_r-element plaintext, and homomorphic multiplication of an input and memory value in R_r is dominated by 4 R_q-multiplications (with correctness if the resulting product y over R maintains $\|y\|_\infty \leq B$). For concrete parameters, the resulting HSS shares will be of comparable size to the analogous SHE-based approach, but will offer significantly cheaper homomorphic multiplication operations—faster by approximately one order of magnitude. (See "Comparison to SHE-based solutions" discussion below.)

We further explore extensions and optimizations to the core construction, within the following settings:

1. *Secret-key HSS*: For applications where all secrets in the system originate from a single party; e.g., the client in 2-server Private Information Retrieval.
2. *Degree-2 computations*: For the special case of homomorphic evaluation of degree-2 polynomials (and extension to low-degree polynomials), with applications to secure computation.
3. *SIMD computations*: Direct support for homomorphic evaluation of "packed" single-instruction multiple data (SIMD) parallel computations on data items encoded as vectors. This has useful application to parallel computations, and to PIR-type applications, where one wishes to perform several identical evaluations on different database items.

Comparison to existing approaches. We briefly discuss our resulting HSS in reference to existing approaches for comparable function classes.

Comparison to Group-Based HSS. Our core construction framework resembles the HSS schemes of Boyle, Gilboa, and Ishai [7] and successors [10, 6, 21], which rely on various flavors of discrete logarithm type assumptions in cryptographically hard Abelian groups (e.g., Decisional Diffie Hellman (DDH) or circular security of ElGamal). We refer to this line as "group-based" HSS.

Despite many algorithmic and heuristic advances, all works in this line are subject to a common computation barrier: In addition to their upper bounds, Dinur, Keller, and Klein [21] showed that (barring a breakthrough in discrete logarithm techniques[4]) performing a homomorphic multiplication via this general approach with plaintext space size B and correctness error δ *requires* runtime $T = \Omega(\sqrt{B/\delta})$. Of particular note, this inherently restricts support to plaintext spaces of polynomial size B, as well as inverse-polynomial error δ.

- *Superpolynomial-size plaintext space.* In contrast, our HSS scheme can directly support operations within superpolynomial plaintext spaces over the ring $\mathbb{Z}[X]/(X^N + 1)$, with complexity growing roughly as the logarithm of the maximum magnitude B. This circumvents blowups associated with artificial emulation of larger input spaces by breaking input elements into small pieces (e.g., bits) and operating piecewise. Such blowups manifest even when operating over small inputs, as encodings of the *secret key* as a plaintext are necessary in order to support homomorphism.
- *Negligible error.* Our HSS also enjoys negligible correctness error. Beyond a theoretical distinction, this greatly affects efficiency. Even to obtain constant failure probability in group-based approaches, homomorphic evaluation of S multiplications requires computation scaling as $S^{3/2}$ [21], since the error of each individual multiplication must be pushed down to $\sim \delta/S$ to reach overall error δ. The presence (or non-presence) of error may further *leak* information about the secret inputs; this adds layers of complexity and overhead to HSS-based applications, wherein effects of error must be sanitized before homomorphically evaluated output shares can be exchanged [7, 10, 6].

[4] Namely, solving the Discrete Logarithm in a Interval problem with interval length R in time $o(\sqrt{R})$.

– *Cheaper operations.* Overall, the resulting lattice-based schemes require cheaper operations than group-based alternatives, e.g. replacing cryptographic group exponentiations by simple polynomial ring multiplications (efficiently implementable using FFT). Most stark in contrast: the expensive "share-conversion" steps in group-based approaches—requiring large scales of repeated group multiplications and pattern matches, and dominating computation costs in homomorphic evaluation—are trivialized given our new techniques.

Comparison to SHE-based solutions. Two-server HSS can also be constructed from threshold variants of somewhat or fully homomorphic encryption, by replacing the (interactive) distributed decryption procedure with the non-interactive "rounding" technique from [22] to give additive shares of the output. Using FHE this can give HSS for circuits, although the computational overhead either grows with the depth of the circuit [14] or is independent of the circuit size but very large due to a costly bootstrapping step.[5]

SHE is reasonably practical for evaluating low-depth circuits, where the dominant cost is homomorphic multiplication. There are several different approaches to SHE multiplication, all of which have various complications due to the need for a "key-switching" procedure [14, 27] to avoid ciphertext expansion, as well as either modulus switching [14] or scale-invariant operations [13, 24] to reduce noise growth.[6]

Our approach avoids these complications, leading to a conceptually simpler and more efficient scheme with several advantages over SHE-based HSS.

– *Cheaper multiplication.* Our homomorphic multiplication procedure is much simpler and cheaper than in SHE, since we avoid costly ciphertext expansion or key/modulus-switching procedures which are inherent in most SHE schemes. Concretely, our multiplication procedure has roughly the cost of *2 decryption operations* in SHE, which we estimate improves our performance by around an order of magnitude based on recent implementation results [31]. While our supported multiplications are of a "restricted" form, requiring one of the multiplicands to be an original input value, this has a mild effect within low-degree computations, which are anyway the competitive regime for SHE.
– *Simpler setup.* Since we do not need key-switching, we also avoid the cost of setting up the key-switching material in a distributed manner, which is a source of additional complexity in threshold FHE [2] as it requires generating several "quasi-encryptions" of s^2, where s is the secret-shared private key.
– *No noise growth.* Unlike FHE, ciphertexts in our homomorphic evaluation procedure do not incur noise growth, which increases ciphertext size and limits the number of homomorphic operations. Instead, we are only limited

[5] Although the cost of bootstrapping has fallen dramatically in recent years [32, 23, 16, 17], the efficiency is still orders of magnitude worse than low-depth somewhat homomorphic encryption using SIMD operations.

[6] So-called "third generation" SHE schemes based on GSW [28] have simpler homomorphic multiplication, but much larger ciphertexts that grow with $\Omega(N \log^2 q)$ instead of $O(N \log q)$, for (R)LWE dimension N and modulus q.

in that the parameters must be chosen based on an upper bound on the maximum size of any plaintext value (without modular reduction) during the computation.

Sample applications. To illustrate the potential of our techniques, we consider two example use-cases of HSS for branching programs. Firstly, we look at secure two-party computation of low-degree polynomials, and its application to generating various forms of correlated randomness. Many MPC protocols use preprocessed, correlated randomness such as Beaver multiplication triples, matrix multiplication triples or truth-table correlations to achieve a very fast "online" protocol. Protocols such as SPDZ [20] often use SHE to generate this randomness, whereas using HSS (considered in [6]) has potential to greatly improve computational costs, and reduce the round complexity to just a single round, while paying a slight overhead with larger ciphertexts. Secondly, we look at 2-server Private Information Retrieval (PIR), which allows a client to perform private queries to a public database. Our HSS for branching programs allows a much richer set of queries than previous, practical schemes based on one-way functions [40], and in this case we can reduce the share size compared with using SHE, as well as the computation.

1.2 Technical Overview

Recall our HSS is with respect to an encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with *nearly linear decryption* over ring $\mathbb{Z}[X]/(X^N + 1)$ (as discussed above), with moduli $r \leq B \ll p \ll q/B$ and parameter d. Ciphertexts and the secret key of the encryption scheme are elements $\mathbf{c}, \mathbf{s} \in R_q^d$ with $\mathbf{s} = (1, \widehat{\mathbf{s}}) \in R_q \times R_q^{d-1}$, the plaintext space of encryption is R_p, and we will support homomorphic operations over R_r for computations for which all intermediate computation values y (as performed over R) remain bounded by $\|y\|_\infty \leq B$. (We will denote $y \in [R]_B$ to highlight that arithmetic is *not* performed modulo B.)

The core of our HSS resembles the DDH-based framework of [7], translated to the setting of lattice-based encryption. The HSS public key pk is precisely the public key of the encryption scheme.[7] The evaluation keys $(\mathsf{ek}_0, \mathsf{ek}_1) \in R_q^d \times R_q^d$ are additive secret shares of the key $\mathbf{s} \in R_q^d$ over R_q.[8] Homomorphic evaluation maintains the invariant that for every intermediate computation value $y \in [R]_B$, Party 0 and Party 1 will hold *additive shares* $(\mathbf{t}_0^y, \mathbf{t}_1^y) \in R_q^d \times R_q^d$ of the product $y \cdot \boldsymbol{s} \in R_q^d$ over R_q. This directly admits homomorphic addition, by locally adding the corresponding secret shares.

As usual, the challenge comes in addressing multiplication. We support homomorphic "restricted" multiplications, between any intermediate computation value y and input value x. To aid this operation, the HSS sharing of input

[7] Note that nearly linear decryption generically implies existence of a public-key encryption procedure.

[8] This can be decreased to $(d-1)$ R_q-elements communicated, as $s_1 = 1 \in R_q$.

$x \in [R]_B$ will be a (componentwise) encryption of $x \cdot \mathbf{s}$: i.e., d ciphertexts $\mathbf{C}^x = (\mathbf{c}^{x \cdot s_1}, \ldots, \mathbf{c}^{x \cdot s_d}) \in (R_q^d)^d$. Interestingly, these encryptions can be generated given just pk of the encryption, leveraging a weak form of key-dependent message (KDM) security implied by nearly linear decryption—see "KDM Security" discussion below. Combining the HSS encoding $\mathbf{C}^x$ of x with the secret shares $(\mathbf{t}_0^y, \mathbf{t}_1^y)$ for y, nearly linear decryption then gives us:

$$\text{for every } i \in [d]: \quad \langle \mathbf{t}_0^y, \mathbf{c}^{x \cdot s_i} \rangle + \langle \mathbf{t}_1^y, \mathbf{c}^{x \cdot s_i} \rangle = \langle y \cdot \mathbf{s}, \mathbf{c}^{x \cdot s_i} \rangle \approx (q/p) \cdot xy \cdot s_i \quad \text{over } R_q.$$

Collectively, this *almost* yields the desired additive shares of $xy \cdot \mathbf{s} \in R_q^d$ to maintain the homomorphic evaluation invariant.

Rounding. Our first observation is that we can use the non-interactive rounding trick as in [22] to locally convert the approximate shares of $(q/p) \cdot xy \cdot s_i$ over R_q from above, to *exact* shares of $xy \cdot s_i$ over R_p. Concretely, each party simply scales his share by (p/q) and *locally* rounds to the nearest integer value. This operation heavily relies on the fact that there are 2 parties, and provides correct output shares over R_p with error probability pB/q, negligible for $p \ll q/B$.

However, this is not quite what we need: the resulting secret shares of $xy \cdot \mathbf{s}$ are over R_p, *not* R_q. This means we cannot use the shares again to "distributively decrypt" the original set of ciphertexts $\mathbf{C}^{x'}$ for any input x', a task whose operations must take place over R_q. (In fact, information about the $s_i \in R_q$ may even be *lost* when taken mod p.) Performing a second analogous multiplication would then necessitate a *second set* of ciphertexts, over a smaller modulus: namely with R_p playing the original role of R_q, and some $p_1 \ll p$ playing the previous role of p. In such fashion, one can devise a *leveled* HSS scheme operating via a sequence of decreasing moduli $q \gg p \gg p_1 \gg p_2 \gg \cdots p_{\text{deg}}$, where each step must drop by a superpolynomial factor to guarantee negligible correctness error. The size and complexity of such HSS scheme, however, would grow significantly with the desired depth of homomorphic computation.

Lifting. We avoid the above conundrum by (quite literally) doing *nothing*. Our observation is as follows. In general, converting secret shares up to a higher modulus constitutes a problem: e.g., even from $\mathbb{Z}_2$ to $\mathbb{Z}_3$ we have $1 + 1 \equiv 0 \in \mathbb{Z}_2$ turning to $1 + 1 \equiv 2 \in \mathbb{Z}_3$. However, if we can guarantee that the secret shared payload is *very small* compared to the modulus, this wraparound problem drops to a negligible fraction of possible secret shares. Concretely, given shares $t_0 + t_1 \equiv t \mod p$ for $t_0, t_1, t \in (-\lfloor p/2 \rceil, \ldots, \lfloor (p-1)/2 \rceil]$, then $t_0 + t_1 = t$ with equality (over $\mathbb{Z}$) unless $t - t_0$ falls $\leq -\lfloor p/2 \rceil$ or $> \lfloor (p-1)/2 \rceil$. If t_0 is randomly chosen and $t \ll p$, then these corner cases occur with only negligible probability. Conditioned on this, conversion to shares modulo q is immediate: it *already* holds that $t_0 + t_1 = t \mod q$.

Recall that we wish to perform share conversion on payload values of the form $y \cdot s_i \in R_q$. To use this trick, we must thus adjust the construction to guarantee that the shares t_b are distributed randomly, and that any such payload value has low magnitude $\|y \cdot s_i\|_\infty \ll p$. Rerandomizing shares is accomplished by each party shifting his share by the same pseudorandom offset (determined via a

PRF). Ensuring low magnitude of $y \cdot s_i$ is done via two pieces. First, we leverage a result of Applebaum *et al.* [1], which allows us to replace a randomly sampled secret key $\mathbf{s} \in R_q^d$ of encryption with one sampled from the *low-magnitude* noise distribution, without loss of security. Second, we introduce an additional modulus level $B \ll p$, and address only computations which remain bounded in magnitude by $\|y\|_\infty \leq B$. Together, this will ensure each value $y \cdot s_i$ has small norm in comparison to p, and thus the shares of $y \cdot s_i$ over R_p can be *directly* interpreted as shares of $y \cdot s_i$ over R_q, successfully returning us to the desired homomorphic evaluation invariant. Finally, using the same low-magnitude-payload share conversion trick, in the final step of computation, the parties can convert their shares of $y \cdot \mathbf{s} = (y, y \cdot \widehat{\mathbf{s}})$ (recall $\mathbf{s} = (1, \widehat{\mathbf{s}})$) over R_q to shares of y over R_r for target output modulus r.

Ultimately, the HSS scheme uses three moduli levels: $B \ll p \ll q/B$. Correctness holds as long as the magnitude of computation values is bounded within $[R]_B$. Homomorphic evaluation maintains secret shares over R_q as its invariant. Each homomorphic multiplication drops down to R_p to remove effects of noise, then steps back up to shares over R_q to reinstate the invariant. An advantageous side effect of this structure is that, conditioned on remaining within magnitude bound B (e.g., Boolean computations), the size of our HSS shares is completely *independent* of the depth or size of the homomorphic computation.

To conclude, we highlight some of the additional ideas and techniques arising within our scheme and extensions.

KDM security. Our HSS reveals encryptions of the form $\mathsf{Enc}(x \cdot s_i)$, for input x and secret key $\mathbf{s}$ of the underlying encryption scheme. Key-dependent message (KDM) security of the encryption scheme with respect to this class of linear functions of $\mathbf{s}$ follows from its nearly linear decryption structure. As typical in KDM literature (following [1, 15]), this is shown by demonstrating as an intermediate step that such encryptions can be efficiently generated from knowledge only of the public key and rerandomized to "look like" fresh encryptions.

In our construction, we *leverage* this efficient generation procedure. This enables pk of the HSS to consist purely of the public key of the encryption scheme, while still allowing parties to encode their respective inputs x as $\{\mathsf{Enc}(x \cdot s_i)\}_{i \in [d]}$. The corresponding encoding procedure is simpler and achieves better parameters than publishing $\{\mathsf{Enc}(s_i)\}_{i \in [d]}$ as part of pk (as was done in [7]), as this introduces extra ciphertexts as well as a second noise term that must be "drowned" by larger noise when scaling by x and rerandomizing.

Secret key as an input. For our RLWE-based HSS schemes with plaintext space $R_r = \mathbb{Z}_r[X]/(X^N + 1)$, and when sampling the secret key $\mathbf{s} = (1, \widehat{s}) \in R_q^2$ from the low-magnitude noise distribution, it holds that $\widehat{s}$ *itself* lies within the supported input space R_r of the HSS. This is implicitly exploited in our attained HSS efficiency, e.g. where our encryptions of $x \cdot \boldsymbol{s} = (x, x\widehat{s}) \in R_q^2$ can consist of just *two* ciphertexts (instead of λ).

However, this also opens *qualitatively* new approaches toward optimization. For example, suppose the evaluation keys $\mathsf{ek}_0, \mathsf{ek}_1$ are augmented with shares of $(\widehat{s})^2$, as well as shares of $\widehat{s}$. We can then view the shares of $(\widehat{s}, (\widehat{s})^2) = \widehat{s}(1, \widehat{s})$ as

HSS sharings of the computation value $\widehat{s}$, and thus use them to homomorphically multiply an input x by $\widehat{s}$. For degree 2, this allows us to save sending encryptions of $x \cdot \widehat{s}$ for inputs x, since we can now generate these "for free" using $\mathsf{ek}_0, \mathsf{ek}_1$ and the encryptions of x, reducing the share size by a factor of two.

SIMD operations. If the underlying encryption scheme is over a ring R of the right form, then our basic HSS supports homomorphic evaluation of "single instruction, multiple data" (SIMD) operations. Namely, suppose $R = \mathbb{Z}[X]/(X^N+1)$ where X^N+1 splits over R_r (for some prime $r \geq 2$) into pairwise different irreducible polynomials of degree $k \in \mathbb{N}$: that is, $R_r \cong \mathbb{F}_{r^k} \times \cdots \times \mathbb{F}_{r^k}$ for N/k copies of $\mathbb{F}_{r^k}$. In such case, our homomorphic additions and multiplications over R_r directly translate to corresponding SIMD operations within the individual computation "slots."

A caveat of this correspondence is that the magnitude bound requirement B over R_r (in *coefficient* embedding) does not translate directly to a per-slot magnitude bound of B (in CRT embedding). Thus current SIMD support can effectively handle low-degree computations, but suffers performance degradation as the degree increases, even if the magnitude of the SIMD computations is bounded. An interesting goal for future investigation will be to devise new ways of packing to mitigate this disadvantage.

Beyond 2 parties. A major open problem is constructing an efficient HSS scheme with additive reconstruction for a reasonably expressive function class in the setting of 3 or more parties. Unfortunately, even for 3 parties, the rounding and lifting techniques we use would result in an error with *constant* probability for any choice of underlying parameters.

The authors of [22] manage to extend their construction of "spooky encryption" to the multiparty setting by using a GMW-like approach [30]. We can apply this to our scheme for degree-2 functions, by performing pairwise products on shares involving pairwise secret keys. However, it is not clear how to extend this further, as one would need to obtain shares of the pairwise outputs under all other secret keys.

2 Preliminaries

We begin this section by introducing some notation. For notation that we consider common knowledge we refer to the full version [12]. We denote our security parameter by λ. Throughout this paper we consider all parameters to implicitly depend on λ, e.g. by $\ell \in \mathbb{N}$ we actually consider ℓ to be a function $\ell\colon \mathbb{N} \to \mathbb{N}$, but simply write ℓ in order to refer to $\ell(\lambda)$.

For a real number $x \in \mathbb{R}$, by $\lfloor x \rceil \in \mathbb{Z}$ we denote the element closest to $x \in \mathbb{R}$, where we round up when the first decimal place of x is 5 or higher.

For $x \in \mathbb{Z}[X]/(X^N+1)$ the maximum norm of x is defined as $\|x\|_\infty := \max |(x_1, \ldots, x_N)|$, where $x_i \in \mathbb{Z}$ such $x = \sum_{i=1}^{N} x_i X^{i-1} \mod X^N + 1$.

For $p \in \mathbb{N}$, by R_p we denote R/pR. Note that we consider R_p as elements for which all coefficients are in the interval $(-\lfloor p/2 \rceil, \ldots, \lfloor (p-1)/2 \rceil]$. For $B \in \mathbb{N}$,

we denote $[R]_B := \{x \in R \mid \|x\|_\infty \leq B\}$. More generally, for an interval $I \subseteq \mathbb{Z}$, we write $R|_I$ to denote all elements of R that have only coefficients in I.

We denote vectors by bold lower-case letters and matrices by bold upper-case letters. We interpret vectors as column-vectors. For a vector $\mathbf{x} \in R^\ell$, by x_i we refer to the i-th entry (for $i \in \{1, \ldots, \ell\}$).

We consider *public-key encryption schemes* that satisfy the security notion of *pseudorandomness of ciphertexts*. For a formal definition we refer to the full version.

2.1 Homomorphic Secret Sharing

We consider homomorphic secret sharing (HSS) as introduced in [7]. By default, in this work, the term HSS refers to a public-key variant of HSS. Unlike [7], we do not need to consider non-negligible δ error failure probability.

Definition 2 (Homomorphic Secret Sharing). *A (2-party, public-key)* Homomorphic Secret Sharing (HSS) *scheme for a class of programs $\mathcal{P}$ over a ring R with input space $\mathcal{I} \subseteq R$ consists of PPT algorithms* $(\mathsf{HSS.Gen}, \mathsf{HSS.Enc}, \mathsf{HSS.Eval})$ *with the following syntax:*

- $\mathsf{HSS.Gen}(1^\lambda)$*: On input a security parameter 1^λ, the key generation algorithm outputs a public key* pk *and a pair of evaluation keys* $(\mathsf{ek}_0, \mathsf{ek}_1)$.
- $\mathsf{HSS.Enc}(\mathsf{pk}, x)$*: Given public key* pk *and secret input value $x \in \mathcal{I}$, the encryption algorithm outputs a ciphertext* ct.
- $\mathsf{HSS.Eval}(b, \mathsf{ek}_b, (\mathsf{ct}^{(1)}, \ldots, \mathsf{ct}^{(\rho)}), P, r)$*: On input party index $b \in \{0,1\}$, evaluation key* ek_b*, vector of ρ ciphertexts, a program $P \in \mathcal{P}$ with ρ input values and an integer $r \geq 2$, the homomorphic evaluation algorithm outputs $y_b \in R_r$, constituting party b's share of an output $y \in R_r$.*

The algorithms $(\mathsf{HSS.Gen}, \mathsf{HSS.Enc}, \mathsf{HSS.Eval})$ *should satisfy the following correctness and security requirements:*

- **Correctness:** *For all $\lambda \in \mathbb{N}$, for all $x^{(1)}, \ldots, x^{(\rho)} \in \mathcal{I}$, for all programs $P \in \mathcal{P}$ with size $|P| \leq \mathsf{poly}(\lambda)$ and $P(x^{(1)}, \ldots, x^{(\rho)}) \neq \perp$, for integer $r \geq 2$, for $(\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{HSS.Gen}(1^\lambda)$ and for $\mathsf{ct}^{(i)} \leftarrow \mathsf{HSS.Enc}(1^\lambda, \mathsf{pk}, x^{(i)})$ we have*

$$\Pr^{\mathsf{cor}}_{\mathsf{HSS},(x^{(i)})_i,P,r}(\lambda) := \Pr\left[y_0 + y_1 = P(x^{(1)}, \ldots, x^{(\rho)}) \mod r\right] \geq 1 - \lambda^{-\omega(1)},$$

  where

$$y_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{ek}_b, (\mathsf{ct}^{(i)})_i, P, r)$$

  for $b \in \{0,1\}$ and where the probability is taken over the random coins of $\mathsf{HSS.Gen}$, $\mathsf{HSS.Enc}$ *and* $\mathsf{HSS.Eval}$.
- **Security:** *For all security parameters $\lambda \in \mathbb{N}$, for all PPT adversaries $\mathcal{A}$ that on input 1^λ output a bit $b \in \{0,1\}$ (specifying which encryption key to*

corrupt), and input values $x_0, x_1 \in \mathcal{I}$, we require the following advantage to be negligible in λ:

$$\mathrm{Adv}^{\mathsf{sec}}_{\mathsf{HSS},\mathcal{A}}(\lambda) := \left| \Pr\left[\mathcal{A}(\mathsf{input}_b) = \beta \;\middle|\; \begin{array}{l} (b, x_0, x_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda), \\ \beta \leftarrow \{0,1\}, \\ (\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{HSS.Gen}(1^\lambda), \\ \mathsf{ct} \leftarrow \mathsf{HSS.Enc}(\mathsf{pk}, x_\beta), \\ \mathsf{input}_b := (\mathsf{state}, \mathsf{pk}, \mathsf{ek}_b, \mathsf{ct}) \end{array} \right] - \frac{1}{2} \right|.$$

Remark 1. Within applications, we additionally consider a secret-key variant of HSS. For details we refer to the full version.

2.2 Computational Models

Our main HSS scheme naturally applies to programs P in a computational model known as *Restricted Multiplication Straight-line (RMS)* programs [18, 7].

Definition 3 (RMS programs). *An RMS program consists of a magnitude bound B_{max} and an arbitrary sequence of the four following instructions, sorted according to a unique identifier $\mathsf{id} \in \mathcal{S}_{\mathsf{id}}$:*

- *Load an input into memory:* $(\mathsf{id}, \widehat{y}_j \leftarrow \widehat{x}_i)$.
- *Add values in memory:* $(\mathsf{id}, \widehat{y}_k \leftarrow \widehat{y}_i + \widehat{y}_j)$.
- *Add input values:* $(\mathsf{id}, \widehat{x}_k \leftarrow \widehat{x}_i + \widehat{x}_j)$.
- *Multiply memory value by input:* $(\mathsf{id}, \widehat{y}_k \leftarrow \widehat{x}_i \cdot \widehat{y}_j)$.
- *Output from memory, as R element:* $(\mathsf{id}, r, \widehat{O}_j \leftarrow \widehat{y}_i)$.

If at any step of execution the size of a memory value exceeds the bound B_{max} (i.e. $\|\widehat{y}_j\|_\infty > B_{\mathsf{max}}$), the output of the program on the corresponding input is defined to be $\perp$. Otherwise the output is the sequence of $\widehat{O}_j$ values, sorted by id*. We define the* size *(resp.,* multiplicative size*) of an RMS program P as the number of instructions (resp., multiplication and load input instructions). Note that we consider addition of input values merely for the purpose of efficiency. We denote the maximum number of additions on input values by $P_{\mathsf{inp}+}$.*

3 HSS from encryption with nearly linear decryption

As explained in the introduction, the core of our HSS construction is an encryption scheme with nearly linear decryption, where nearly linear means that for message $m \in R_p := R/pR$, secret key $\mathbf{s} \in R_q^d$, and ciphertext $\mathbf{c} \in R_q^d$ encrypting m, for some "small" noise $e \in R$ we have

$$\langle \mathbf{s}, \mathbf{c} \rangle = (q/p) \cdot m + e \mod q.$$

We begin in Section 3.1 by explaining our two main share conversion tricks, which allow two parties holding secret shares of $(q/p) \cdot m + e \mod q$ for small m to locally modify their values, such that in the end each party holds a secret share of the message m *modulo* q. In Section 3.2 we present our formal definition of nearly linear decryption, and prove two properties that it implies. Then, in Section 3.3 we give our HSS construction based on any such encryption scheme.

3.1 Computation on 2-party secret shared values

First, we present a local rounding trick as in [22] which allows to recover the shares of m (modulo p). The idea is that if q/p is large, the probability that the error term e leads to a rounding error is small. Note that it is crucial here that we are in the 2-party setting, where the secret shares of $(q/p) \cdot m + e \mod q$ have both approximately (that is, except for the error e) the same distance from some multiple of q/p. In fact, even for arbitrarily large gap between p and q, rounding for 3 or more parties fails with constant probability. For a proof of the rounding lemma we refer to the full version.

Lemma 1 (Rounding of noisy shares). *Let $p, q \in \mathbb{N}$ be modulus values with $q/p \geq \lambda^{\omega(1)}$. Let $R = \mathbb{Z}[X]/(X^N + 1)$ for N a power of 2 (i.e. $N = 2^n$ for $n \in \mathbb{N}_0$). Let $t_0, t_1 \in R_q$ random subject to*

$$t_0 + t_1 = (q/p) \cdot m + e \mod q$$

for some $m \in R_p, e \in R$ with $q/(p \cdot \|e\|_\infty) \geq \lambda^{\omega(1)}$. Then, for the the deterministic polynomial time procedure Round *that on input $t_b \in R_q$ outputs*

$$\lfloor (p/q) \cdot t_b \rceil \mod p \in R_p$$

it holds:

$$\mathsf{Round}(t_0) + \mathsf{Round}(t_1) = m \mod p$$

with probability at least $1 - N \cdot (\|e\|_\infty + 1) \cdot p/q \geq 1 - \lambda^{-\omega(1)}$ over the choice of the shares t_0, t_1.

The following simple observation constitutes a crucial step of our HSS construction, as it will allow to have several levels of multiplication without requiring a sequence of decreasing moduli. While in general the conversion of secret shares from one modulus to another constitutes a problem, we observe that whenever the secret shared value is *small* in comparison to the modulus, and we use the centered representation of R_p with coefficients in $(-\lfloor p/2 \rceil, \ldots, \lfloor (p-1)/2 \rceil]$, then with high probability the secret sharing actually constitutes a secret sharing over R, so switching to an arbitrary modulus is trivial. Note that (as for rounding) this only holds true in the 2-party setting. For the proof we refer to the full version.

Lemma 2 (Lifting the modulus of shares). *Let $p \in \mathbb{N}$ be a modulus with $p \geq \lambda^{\omega(1)}$. Let $R = \mathbb{Z}[X]/(X^N+1)$ for N a power of 2. Let $m \in R$ and $z_0, z_1 \in R_p$ be random, subject to*

$$z_0 + z_1 = m \mod p.$$

Then, we have

$$z_0 + z_1 = m \text{ over } R$$

with probability at least $1 - (N \cdot (\|m\|_\infty + 1)/p) \geq 1 - \lambda^{-\omega(1)}$ over the choice of the shares z_0, z_1.

3.2 Encryption with nearly linear decryption

We now formally introduce encryption with nearly linear decryption. Basically, we require the following properties: First, there is a way to encrypt certain key-dependent messages without knowledge of the secret key. Second, it is possible to "distributively decrypt" a ciphertext. More precisely, given an encryption of message m and secret shares of some multiple x of the secret key $\mathbf{s}$, there is a way to obtain secret shares of $x \cdot m$ *over the same modulus as the original secret shares*. These properties together enable us to perform several stages of distributed decryption. That is, given an encryption of $x \cdot \mathbf{s}$ (for some value x) and a secret share of $x' \cdot \mathbf{s}$ modulo q, distributed decryption results in a secret share of $x \cdot x' \cdot \mathbf{s}$ modulo q, which can serve as input to another distributed decryption. One way to achieve both properties at once is to require nearly linear decryption.

Definition 4 (Encryption scheme with nearly linear decryption). *Let* $\mathsf{PKE} := (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be a public-key encryption scheme with pseudorandom ciphertexts. We say that* PKE *is a public-key* encryption scheme with nearly linear decryption *if it further satisfies the following properties:*

- ***Parameters:*** *The scheme is parametrized by modulus values $p, q \in \mathbb{N}$, dimension $d \in \mathbb{N}$, and bounds $B_{\mathsf{sk}}, B_{\mathsf{ct}} \in \mathbb{N}$, where $p|q$, $p \geq \lambda^{\omega(1)}$, $q/p \geq \lambda^{\omega(1)}$ and $d, B_{\mathsf{sk}}, B_{\mathsf{ct}} \leq \mathsf{poly}(\lambda)$, as well as a ring $R = \mathbb{Z}[X]/(X^N + 1)$, where $N \leq \mathsf{poly}(\lambda)$ is a power of 2.*[9]
- ***Message space and secret key:*** *The scheme has message space $\mathcal{M} := R_p := R/pR$ and ciphertext space $\mathcal{C} := R_q^d := (R/qR)^d$. The secret key $\mathbf{s}$ returned by $\mathsf{PKE.Gen}$ on input 1^λ is an element of R^d satisfying $\|\mathbf{s}\|_\infty \leq B_{\mathsf{sk}}$. Further, $\mathbf{s}$ is of the form $(1, \widehat{\mathbf{s}})$ for some $\widehat{\mathbf{s}} \in R_p^{d-1}$.*
- ***Nearly linear decryption:*** *For any $\lambda \in \mathbb{N}$, for any $(\mathsf{pk}, \mathbf{s})$ in the image of $\mathsf{Gen}(1^\lambda)$, for any message $m \in R_p$ and for any ciphertext $\mathbf{c} \in R_q^d$ in the image of $\mathsf{PKE.Enc}(\mathsf{pk}, m)$, for some $e \in R$ with $\|e\|_\infty \leq B_{\mathsf{ct}}$ it holds*

$$\langle \mathbf{s}, \mathbf{c} \rangle = (q/p) \cdot m + e \mod q.$$

Notation. *For $(\mathsf{pk}, \mathbf{s}) \leftarrow \mathsf{Gen}(1^\lambda)$ and $\mathbf{m} = (m_1, \ldots, m_d) \in R_p^d$, we denote by $\mathsf{PKE}.\mathbf{Enc}(\mathsf{pk}, \mathbf{m})$ the componentwise encryption $\mathbf{C} \leftarrow (\mathsf{PKE.Enc}(\mathsf{pk}, m_1), \ldots, \mathsf{PKE.Enc}(\mathsf{pk}, m_d))$; we denote by $\mathbf{Dec}(\mathsf{sk}, \mathbf{C})$ the decryption $(\mathsf{PKE.Dec}(\mathsf{sk}, \mathbf{c}_1), \ldots, \mathsf{PKE.Dec}(\mathsf{sk}, \mathbf{c}_d)) \in R_p^d$ of the matrix of d ciphertexts $\mathbf{C} = (\mathbf{c}_1 | \ldots | \mathbf{c}_d) \in R_q^{d \times d}$.*

Remark 2. Encryption with nearly linear decryption can be instantiated based on LWE (e.g. with [37, 1], where $d = \lambda$) and based on RLWE (e.g. with [36, 15], where $d = 2$). Further, it can be instantiated with schemes based on assumptions like module-LWE [34] and LWR [3]. For more details on the instantiation from the RLWE-based encryption scheme of LPR [36], we refer to Section 4, and for the instantiation from the LWE-based encryption scheme of Regev [37] we refer to the full version.

[9] To simplify the analysis, we restrict the definition to 2-power cyclotomic rings. However, our construction can be generalized to arbitrary cyclotomics.

$\underline{\mathrm{Exp}^{\mathsf{kdm-ind}}_{\mathsf{PKE.OKDM},\mathcal{A}}(\lambda):}$	$\underline{\mathcal{O}_{\mathsf{KDM}}(x,j):}$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$	**if** $\beta = 0$
$\beta \leftarrow \{0,1\}$	**return** $\mathsf{PKE.OKDM}(\mathsf{pk},x,j)$
$\beta' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KDM}}(\cdot,\cdot)}(1^\lambda,\mathsf{pk})$	**else**
if $\beta = \beta'$ **return** 1	**return** $\mathsf{PKE.Enc}(\mathsf{pk},0)$
else return 0	

Fig. 1: Security challenge experiment for the KDM oracle.

We prove that our two desired properties are satisfied by any encryption scheme with nearly linear decryption. The first property allows anyone to compute an encryption of any linear function of the secret key without having access to the secret key itself, serving as a "KDM oracle." A similar notion, but for secret-key encryption schemes and with deterministic procedure, was introduced in [4]. For the proof of the following lemma we refer to the full version.

Lemma 3 (KDM oracle). *Let* $\mathsf{PKE} := (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be a public-key encryption scheme with nearly linear decryption and parameters* $(p, q, d, B_{\mathsf{sk}}, B_{\mathsf{ct}}, R)$.

Then, for the PPT procedure $\mathsf{PKE.OKDM}$ *that on input of a public key* pk*, a value* $x \in R$ *and an index* $j \in \{1,\ldots,d\}$ *computes an encryption* $\mathbf{c} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk},0)$ *and outputs*

$$\mathbf{c}_j := (q/p) \cdot x \cdot \mathbf{e}_j + \mathbf{c} \mod q,$$

where $\mathbf{e}_j \in R_q^d$ *is the* j*-th unit vector, the following properties are satisfied.*

- ***Nearly linear decryption to the message*** $x \cdot s_j$***:*** *For any* $\lambda \in \mathbb{N}$*, for any* $(\mathsf{pk},\mathbf{s})$ *in the image of* $\mathsf{Gen}(1^\lambda)$*, and for any ciphertext* $\mathbf{c}_j \in R_q^d$ *in the image of* $\mathsf{PKE.OKDM}(\mathsf{pk},x,j)$*, it holds*

$$\langle \mathbf{s}, \mathbf{c}_j \rangle = (q/p) \cdot (x \cdot s_j) + e \mod q$$

  for some $e \in R$ *with* $\|e\|_\infty \leq B_{\mathsf{ct}}$.
- ***Security:*** *For any* $\lambda \in \mathbb{N}$ *and any PPT adversary* $\mathcal{A}$ *we have that*

$$\mathrm{Adv}^{\mathsf{kdm-ind}}_{\mathsf{PKE.OKDM},\mathcal{A}}(\lambda) := \left| \Pr\left[\mathrm{Exp}^{\mathsf{kdm-ind}}_{\mathsf{PKE.OKDM},\mathcal{A}}(\lambda) = 1\right] - 1/2 \right|$$

  is negligible in λ*, where* $\mathrm{Exp}^{\mathsf{kdm-ind}}_{\mathsf{PKE.OKDM},\mathcal{A}}(\lambda)$ *is as defined in Figure 1*

By $\mathsf{PKE.}\mathbf{OKDM}(\mathsf{pk},x)$ *we denote the KDM oracle that returns a componentwise encryption of* $x \cdot \mathbf{s}$*, i.e. that outputs the matrix* $(\mathsf{PKE.OKDM}(\mathsf{pk},x,1),\ldots,\mathsf{PKE.OKDM}(\mathsf{pk},x,d)) \in R_q^{d\times d}$.

The following shows that any encryption with nearly linear decryption allows two parties to perform decryption *distributively*, employing their respective shares of the secret key to obtain a secret share of the corresponding message *modulo* q. Further, the scheme inherently supports homomorphic addition of ciphertexts, and the distributed decryption property holds accordingly for any sum of a bounded number of ciphertexts (generated from Enc or OKDM).

Lemma 4 (Distributed decryption of sums of ciphertexts). *Let* $\mathsf{PKE} := (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be a public-key encryption scheme with nearly linear decryption and parameters* $(p, q, d, B_{\mathsf{sk}}, B_{\mathsf{ct}}, R)$*, where* R *has dimension* N*. Let* $\mathsf{PKE.OKDM}$ *be the KDM oracle from Lemma 3. Let* $B_{\mathsf{add}} \in \mathbb{N}$ *with* $B_{\mathsf{add}} \leq \mathsf{poly}(\lambda)$*. Then the deterministic polynomial time decryption procedure* $\mathsf{PKE.DDec}$ *that on input* $b \in \{0,1\}, \mathbf{t}_b \in R^d, \mathbf{c} \in R_q^d$ *outputs*

$$\mathsf{Round}(\langle \mathbf{t}_b, \mathbf{c} \rangle \mod q) \in R_q$$

(where Round *is as in Lemma 1) satisfies the following:*

For all $x \in R_p$ *with* $p/\|x\|_\infty \geq \lambda^{\omega(1)}$ *and* $q/(p \cdot \|x\|_\infty) \geq \lambda^{\omega(1)}$*, for all* $(\mathsf{pk}, \mathbf{s}) \leftarrow \mathsf{Gen}(1^\lambda)$*, for all messages* $m_1 \ldots, m_{B_{\mathsf{add}}} \in R_p$*, for all encryptions* $\mathbf{c}_i$ *of* m_i *that are either output of* $\mathsf{PKE.Enc}$ *or of* $\mathsf{PKE.OKDM}$ *(in that case we have* $m_i = x_i \cdot s_j$ *for some value* $x_i \in R_p$ *and some index* $j \in \{1, \ldots, d\}$*) and for shares* $\mathbf{t}_0, \mathbf{t}_1 \in R_q^d$ *random subject to*

$$\mathbf{t}_0 + \mathbf{t}_1 = x \cdot \mathbf{s} \mod q$$

for $\mathbf{c} := \sum_{i=1}^{B_{\mathsf{add}}} \mathbf{c}_i$ *and* $m := \sum_{i=1}^{B_{\mathsf{add}}} m_i$ *it holds*

$$\mathsf{PKE.DDec}(0, \mathbf{t}_0, \mathbf{c}) + \mathsf{PKE.DDec}(1, \mathbf{t}_1, \mathbf{c}) = x \cdot m \mod q$$

with probability over the random choice of the shares $\mathbf{t}_0, \mathbf{t}_1$ *of at least*

$$1 - N \cdot (N \cdot B_{\mathsf{add}} \cdot \|x\|_\infty \cdot B_{\mathsf{ct}} \cdot p/q + \|x \cdot m\|_\infty / p + p/q + 1/p) \geq 1 - \lambda^{-\omega(1)}.$$

For $\mathbf{C} = (\mathbf{c}_1 | \ldots | \mathbf{c}_d) \in R_p^{d \times d}$ *by* $\mathbf{m} \leftarrow \mathsf{PKE.}\mathbf{DDec}(b, \mathbf{t}_b, \mathbf{C})$ *we denote the componentwise decryption* $\mathbf{m} \leftarrow (\mathsf{PKE.DDec}(b, \mathbf{t}_b, \mathbf{c}_1), \ldots, \mathsf{PKE.DDec}(b, \mathbf{t}_b, \mathbf{c}_d)) \in R_p^d$*.*

For a proof that $\mathsf{PKE.DDec}$ indeed satisfies the required we refer to the full version. The idea is that nearly linear decryption allows (almost) homomorphic addition of ciphertexts with linear growth in the error. As $q/(p \cdot \|x\|_\infty) \geq \lambda^{\omega(1)}$ and the vectors $\mathbf{t}_b$ are individually random, by Lemma 1 we can recover $x \cdot m \mod p$ with overwhelming probability. Finally, as $p \geq \lambda^{\omega(1)}$, by Lemma 2 we can *lift* the modulus q (as with overwhelming probability the shares constitute a correct sharing of $x \cdot m$ over R and thus R_q).

Remark 3. Note that our techniques also extend to encryption schemes which encrypt messages in low-order symbols, e.g. where $\langle \mathbf{s}, \mathbf{c} \rangle = m + p \cdot e \mod q$ for p and q coprime. For more details we refer to the full version.

3.3 HSS from encryption with nearly linear decryption

We now present our construction of a public-key HSS from an encryption scheme with nearly linear decryption. For various extensions that allow to improve the efficiency in specific applications, we refer to Section 3.4.

$\underline{\mathsf{HSS.Gen}(1^\lambda):}$

- Generate a key pair $(\mathsf{pk}, \mathbf{s}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$ for encryption and draw a PRF key $K \xleftarrow{\$} \mathcal{K}$. //Recall $\mathbf{s} = (1, \widehat{\mathbf{s}}) \in R_q \times R_q^{d-1}$.
- **Secret share the secret key.** Choose $\mathbf{s}_0 \xleftarrow{\$} R_q^d$ at random. Define

$$\mathbf{s}_1 := \mathbf{s} - \mathbf{s}_0 \mod q.$$

- Output pk and $\mathsf{ek}_b \leftarrow (K, \mathbf{s}_b)$.

$\underline{\mathsf{HSS.Enc}(1^\lambda, \mathsf{pk}, x):}$

- **Encrypt the input.** Compute and output $\mathbf{C}^x \leftarrow \mathsf{PKE.}\mathbf{OKDM}(\mathsf{pk}, x)$. //This corresponds to $\mathbf{C}^x = \mathsf{PKE.}\mathbf{Enc}(\mathsf{pk}, x \cdot \mathbf{s}) \in R_q^{d \times d}$.

$\underline{\mathsf{HSS.Eval}(b, \mathsf{ek}_b, (\mathbf{C}^{x^{(1)}}, \ldots, \mathbf{C}^{x^{(\rho)}}), P, r):}$

Parse $(K, \mathbf{s}_b) =: \mathsf{ek}_b$, parse P as a sequence of RMS operations and proceed as follows.

- **Load an input into memory:** On instruction $(\mathsf{id}, \mathbf{C}^x)$ compute

$$\mathbf{t}_b^x := \mathsf{PKE.}\mathbf{DDec}(b, \mathbf{s}_b, \mathbf{C}^x) + (1-2b) \cdot \mathsf{PRF}(K, \mathsf{id}) \mod q.$$

- **Add values in memory:** On instruction $(\mathsf{id}, \mathbf{t}_b^x, \mathbf{t}_b^{x'})$ compute

$$\mathbf{t}_b^{x+x'} \leftarrow \mathbf{t}_b^x + \mathbf{t}_b^{x'} + (1-2b) \cdot \mathsf{PRF}(K, \mathsf{id}) \mod q.$$

- **Add input values:** On instruction $(\mathsf{id}, \mathbf{C}^x, \mathbf{C}^{x'})$ compute

$$\mathbf{C}^{x+x'} \leftarrow \mathbf{C}^x + \mathbf{C}^{x'} \mod q.$$

- **Multiply memory value by input:** On instruction $(\mathsf{id}, \mathbf{t}_b^x, \mathbf{C}^{x'})$ compute

$$\mathbf{t}_b^{x \cdot x'} := \mathsf{PKE.}\mathbf{DDec}(b, \mathbf{t}_b^x, \mathbf{C}^{x'}) + (1-2b) \cdot \mathsf{PRF}(K, \mathsf{id}) \mod q.$$

- **Output from memory, as element in R_r:** On instruction $(\mathsf{id}, \mathbf{t}_b^x)$ parse $\mathbf{t}_b^x =: (x_b, \widehat{\mathbf{t}}_b^x)$ for some $x_b \in R_q$, $\widehat{\mathbf{t}}_b^x \in R_q^{d-1}$ and output

$$x_b \mod r.$$

Fig. 2: 2-party public-key homomorphic secret sharing scheme HSS for the class of RMS programs from encryption with nearly linear decryption. Here, $x \in R$ with $\|x\|_\infty \leq B_{\mathsf{inp}}$ is an input value. Throughout, *input values* $x \in R$ are represented by encryptions $\mathbf{C}^x$ of $x \cdot \mathbf{s}$ and *memory values* $x \in R$ are represented by shares $(\mathbf{t}_0^x, \mathbf{t}_1^x) \in R_q^d \times R_q^d$ with $\mathbf{t}_0^x + \mathbf{t}_1^x = x \cdot \mathbf{s} \mod q$.

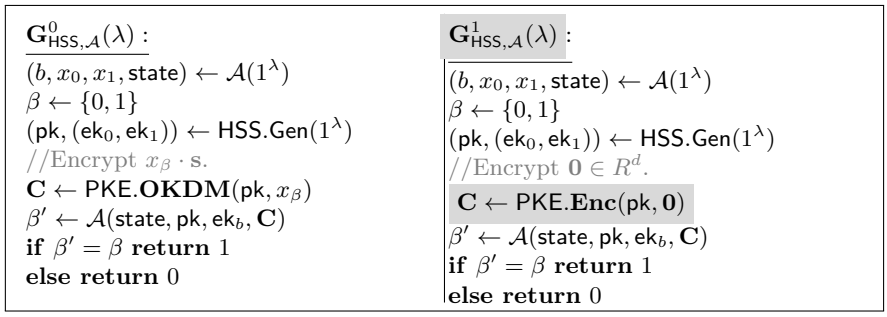

Fig. 3: Games $\mathbf{G}^0_{\mathsf{HSS},\mathcal{A}}(\lambda)$ and $\mathbf{G}^1_{\mathsf{HSS},\mathcal{A}}(\lambda)$ in the proof of Theorem 2 (Sec. of HSS).

Theorem 2 (HSS from encryption with nearly linear decryption). *Let* $\mathsf{PKE} := (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be a secure public-key encryption scheme with nearly linear decryption and parameters* $(p, q, d, B_{\mathsf{sk}}, B_{\mathsf{ct}}, R)$.

- *Let* $B_{\mathsf{inp}} \in \mathbb{N}$ *with* $p/B_{\mathsf{inp}} \geq \lambda^{\omega(1)}$ *and* $q/(B_{\mathsf{inp}} \cdot p) \geq \lambda^{\omega(1)}$.
- *Let* PKE.OKDM *be the KDM oracle from Lemma 3.*
- *Let* PKE.DDec *be the distributed decryption from Lemma 4.*
- *Let* $\mathsf{PRF}\colon \mathcal{K} \times \mathcal{S}_{\mathsf{id}} \to R_q^d$ *be a pseudorandom function.*

Then, the scheme $\mathsf{HSS} = (\mathsf{HSS.Gen}, \mathsf{HSS.Enc}, \mathsf{HSS.Eval})$ *given in Figure 2 is a* 2*-party public-key homomorphic secret sharing scheme with input space* $[R]_{B_{\mathsf{inp}}}$ *for the class of RMS programs with magnitude bound* B_{max}*, where* $p/B_{\mathsf{max}} \geq \lambda^{\omega(1)}$ *and* $q/(B_{\mathsf{max}} \cdot p) \geq \lambda^{\omega(1)}$*. More precisely,* HSS *satisfies the following.*

- ***Correctness:*** *For any* $\lambda \in \mathbb{N}$*, for any* $x^{(1)}, \ldots, x^{(\rho)} \in [R]_{B_{\mathsf{inp}}}$*, for any polynomial-sized RMS program* P *with* $P(x^{(1)}, \ldots, x^{(\rho)}) \neq \perp$ *and magnitude bound* B_{max} *with* $p/B_{\mathsf{max}} \geq \lambda^{\omega(1)}$ *and* $q/(B_{\mathsf{max}} \cdot p) \geq \lambda^{\omega(1)}$*, and for any integer* $r \geq 2$*, there exist a PPT adversary* $\mathcal{B}$ *on the pseudorandomness of* PRF *such that*

$$\Pr^{\mathsf{cor}}_{\mathsf{HSS},(x^{(i)})_i,P,r}(\lambda) \geq 1 - \left(\mathrm{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{B}}(\lambda) + \lambda^{-\omega(1)}\right).$$

- ***Security:*** *For every PPT adversary* $\mathcal{A}$ *on the security of* HSS*, there exists an PPT adversary* $\mathcal{B}$ *on the security of* PKE.OKDM *such that*

$$\mathrm{Adv}^{\mathsf{sec}}_{\mathsf{HSS},\mathcal{A}}(\lambda) \leq \mathrm{Adv}^{\mathsf{kdm-ind}}_{\mathsf{PKE.OKDM},\mathcal{B}}(\lambda).$$

We prove correctness in the following lemma and refer to the full version for the proof of security.

Lemma 5 (Correctness of the HSS). *Let* HSS *be the HSS from Figure 2 with underlying ring* $R = \mathbb{Z}[X]/(X^N + 1)$*. Then, for all* $\lambda \in \mathbb{N}$*, for all inputs* $x^{(1)}, \ldots, x^{(\rho)} \in [R]_{B_{\mathsf{inp}}}$*, for all RMS programs* P*, s.t.*

- P *is of size* $|P| \leq \mathsf{poly}(\lambda)$

– *P has magnitude bound B_{max} with $p/B_{\mathsf{max}} \geq \lambda^{\omega(1)}$ and $q/(B_{\mathsf{max}} \cdot p) \geq \lambda^{\omega(1)}$,*
– *P has maximum number of input addition instructions $P_{\mathsf{inp}+}$*

for $(\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{HSS.Gen}(1^\lambda)$, for $\mathbf{C}^{x^{(i)}} \leftarrow \mathsf{HSS.Enc}(1^\lambda, \mathsf{pk}, x^{(i)})$, there exists an PPT adversary $\mathcal{B}$ on the pseudorandom function PRF with such that correctness holds with probability at least

$$\begin{aligned}\Pr^{\mathsf{cor}}_{\mathsf{HSS},(x^{(i)})_i,P}(\lambda) \geq 1 &- \mathrm{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{B}}(\lambda) - N \cdot (B_{\mathsf{max}} + 1)/q \\ &- |P| \cdot d \cdot N^2 \cdot P_{\mathsf{inp}+} \cdot B_{\mathsf{max}} \cdot (B_{\mathsf{ct}} \cdot p/q + B_{\mathsf{sk}}/p)\,. \\ &- |P| \cdot d \cdot N \cdot (p/q + 1/p).\end{aligned}$$

Proof. We prove correctness via a hybrid argument. Let $\varepsilon_0 := \Pr^{\mathsf{cor}}_{\mathsf{HSS},(x^{(i)})_i,P,r}(\lambda)$. Recall that by ε^0 we denote the probability that homomorphic evaluation of a program P on input $(x^{(1)}, \ldots, x^{(\rho)}) \in [R]^{\rho}_{B_{\mathsf{inp}}}$ employing our HSS presented in Figure 2 is successful (over the random choices of $\mathsf{HSS.Gen}, \mathsf{HSS.Enc}$). Our goal is to prove that for all $x^{(1)}, \ldots, x^{(\rho)} \in [R]_{B_{\mathsf{inp}}}$ and for all bounded RMS programs P the probability ε_0 is negligible in λ.

To this end, let $\varepsilon_1 := \Pr^{1}_{\mathsf{HSS},(x^{(i)})_i,P,r}(\lambda)$ denote the probability that evaluation yields the correct output, where we replace every evaluation of the PRF by inserting a value $\mathbf{r} \xleftarrow{\$} R_q^d$ chosen at random. We show that if the probabilities ε_0 and ε_1 differ significantly, then there exists an adversary $\mathcal{B}$ attacking the underlying PRF PRF. Namely, $\mathcal{B}$ homomorphically evaluates the program P on input $(x^{(1)}, \ldots, x^{(\rho)})$, but instead of evaluating $\mathsf{PRF}(K, \mathsf{id})$ the adversary $\mathcal{B}$ queries its PRF oracle. Finally, $\mathcal{B}$ returns *real* if homomorphic evaluation does not yield the correct result, and *random* otherwise. This yields

$$|\varepsilon_0 - \varepsilon_1| \leq \mathrm{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{B}}(\lambda).$$

It is left to give a lower bound for the probability ε_1. To that end, we prove that with overwhelming probability over the choice of $\mathbf{r} \leftarrow R_q^d$ (in place of the PRF evaluation) all shares $(\mathbf{t}_0^x, \mathbf{t}_1^x)$ computed during homomorphic evaluation of P satisfy

$$\mathbf{t}_0^x + \mathbf{t}_1^x = x \cdot \mathbf{s} = (x, x \cdot \widehat{\mathbf{s}}) \mod q \tag{1}$$

if the function evaluation of P at point $(\mathbf{t}_0^x, \mathbf{t}_1^x)$ corresponds to $x \in R$, where $\mathbf{s} = (1, \widehat{\mathbf{s}}) \in R \times R^{d-1}$ is the secret key returned by $\mathsf{PKE.Gen}$ on input 1^λ. Further, we have that $(\mathbf{t}_0^x, \mathbf{t}_1^x)$ are distributed uniformly at random conditioned on Equation 1.

Assuming Equation 1 is true, by Lemma 2 we have $x_0 + x_1 = x$ over R (and thus over R_r) with probability at least $1 - N \cdot (B_{\mathsf{max}} + 1)/q$.

It is left to prove that indeed Equation 1 holds true during homomorphic evaluation of P except with negligible probability. Recall that $\mathsf{PKE.DDec}$ is the procedure for distributed decryption from Lemma 4. First, assume that distributed decryption is always successful. In this case we prove that any instruction preserves correctness. Note that we do not need to consider the addition of input values and the output of a memory value, as those do not affect the shares.

– **Load an input into memory:** Consider intruction $(\mathsf{id}, \mathbf{C}^x)$ for $b \in \{0,1\}$. Assuming correctness of distributed decryption it holds

$$\begin{aligned}\mathbf{t}_0^x + \mathbf{t}_1^x =& \mathsf{PKE.}\mathbf{DDec}(0, \mathbf{s}_0, \mathbf{C}^x) + \mathbf{r} + \mathsf{PKE.}\mathbf{DDec}(1, \mathbf{s}_1, \mathbf{C}^x) - \mathbf{r} \mod q \\ =& 1 \cdot (x \cdot \mathbf{s}) \mod q = x \cdot \mathbf{s} \mod q.\end{aligned}$$

– **Add values in memory:** Assuming correctness holds for shares $(\mathbf{t}_0^x, \mathbf{t}_1^x)$ and $(\mathbf{t}_0^{x'}, \mathbf{t}_1^{x'})$ we have, as required,

$$\begin{aligned}\mathbf{t}_0^{x+x'} + \mathbf{t}_1^{x+x'} &= \mathbf{t}_0^x + \mathbf{t}_0^{x'} + \mathbf{r} + \mathbf{t}_1^x + \mathbf{t}_1^{x'} - \mathbf{r} \mod q \\ &= x \cdot \mathbf{s} + x' \cdot \mathbf{s} \mod q = (x + x') \cdot \mathbf{s} \mod q.\end{aligned}$$

– **Multiply memory value by input:** Assuming correctness holds for the share $(\mathbf{t}_0^x, \mathbf{t}_1^x)$ and assuming correctness of distributed decryption it holds

$$\begin{aligned}\mathbf{t}_0^{x \cdot x'} + \mathbf{t}_1^{x \cdot x'} =& \mathsf{PKE.}\mathbf{DDec}(0, \mathbf{t}_0^x, \mathbf{C}^{x'}) + \mathsf{PKE.}\mathbf{DDec}(1, \mathbf{t}_1^x, \mathbf{C}^{x'}) \mod q \\ =& x \cdot (x' \cdot \mathbf{s}) \mod q = (x \cdot x') \cdot \mathbf{s} \mod q.\end{aligned}$$

As $\mathbf{r}$ is chosen at random, the distribution of $(\mathbf{t}_0^y, \mathbf{t}_1^y) \in R_q^d$ for $y \in \{x, x + x', x \cdot x'\}$ is random conditioned on Equation 1.

It is left to bound the probability that distributed decryption fails. As for all x computed throughout the evaluation of program P the distribution of $(\mathbf{t}_0^x, \mathbf{t}_1^x) \in R_q^d$ is random conditioned on Equation 1, by Lemma 4 for all messages $m_1 \ldots, m_{P_{\mathsf{inp}+}} \in R_p$ and for all encryptions $\mathbf{c}_i$ of m_i that are output of $\mathsf{PKE.OKDM}$ distributed decryption of $\sum_{i=1}^{P_{\mathsf{inp}+}} \mathbf{c}_i$ fails with probability at most

$$N^2 \cdot P_{\mathsf{inp}+} \cdot \|x\|_\infty \cdot B_{\mathsf{ct}} \cdot p/q + N \cdot \|x \cdot m\|_\infty / p + N \cdot (p/q + 1/p),$$

where $m := \sum_{i=1}^{P_{\mathsf{inp}+}} m_i$. Throughout the evaluation of P we are guaranteed $\|x\|_\infty \leq B_{\mathsf{max}}$ for all intermediary values $x \in R$. Further, for the messages $m_i = x_i \cdot s_{j_i}$ corresponding to outputs of $\mathsf{PKE.OKDM}$ we have

$$\|x \cdot \sum_{i=1}^{P_{\mathsf{inp}+}} x_i \cdot s_{j_i}\|_\infty \leq \sum_{i=1}^{P_{\mathsf{inp}+}} \|x \cdot x_i \cdot s_{j_i}\|_\infty \leq P_{\mathsf{inp}+} \cdot N \cdot B_{\mathsf{max}} \cdot B_{\mathsf{sk}}.$$

Finally, applying a union bound over all $|P| \cdot d$ decryptions yields

$$\begin{aligned}\varepsilon_1 \geq 1 &- N \cdot (B_{\mathsf{max}} + 1)/q - |P| \cdot d \cdot N^2 \cdot P_{\mathsf{inp}+} \cdot B_{\mathsf{max}} \cdot (B_{\mathsf{ct}} \cdot p/q + B_{\mathsf{sk}}/p) \\ &- |P| \cdot d \cdot N \cdot (p/q + 1/p).\end{aligned}$$

3.4 Extensions

In the following we briefly describe some extensions which are tailored to special applications and improve the HSS construction introduced in the previous

section in terms of efficiency. For a complete treatment, we refer the reader to the full version.

Secret-key HSS. For certain applications, where all secret inputs originate from a single party, it is sufficient to consider a *secret-key* HSS. This allows a more efficient instantiation for two reasons. First, the underlying encryption scheme is not required to support ciphertexts from a KDM oracle (but has to be KDM secure), which slightly saves in noise parameters. Further, we can save in terms of computations (at the cost of a larger share size), by replacing the DDec steps for loading an input x into memory, by instead sending the secret shares of $x \cdot \mathbf{s}$ as an additional part of the HSS share.

HSS for degree-2 polynomials. For the restricted class of degree-2 polynomials, we can achieve improved efficiency in both the secret-key and public-key setting, by leveraging the fact that our HSS need only support terminal multiplications.

For the secret-key case, as we do not need to load inputs, we actually only need one level of distributed decryption. This has two advantages: First, it suffices to encrypt $x \in R_p$ instead of $x \cdot \mathbf{s} \in R_p^d$, as the output is not required to allow another distributed encryption. Second, for the same reason, we do not need to lift the modulus of the output of the distributed decryption back to q. Thus, we can choose $p \leq \mathsf{poly}(\lambda)$ and $q \geq \lambda^{\omega(1)}$ (as we no longer must apply Lemma 2).

The idea of our public-key HSS is to change the way inputs are loaded into memory. The idea is to obtain the shares of $x \cdot \mathbf{s} = (x, x \cdot s_2, \ldots, x \cdot s_d) \in R^d$ by decrypting $\mathsf{PKE.Enc}(\mathsf{pk}, x)$ with $\mathbf{s}$ *and* with $s_2 \cdot \mathbf{s}, \ldots, s_d \cdot \mathbf{s}$. This strategy requires a quadratic number of secret shares (namely shares of $\mathbf{s} \cdot \mathbf{s}^\top$), but reduces the number of required encryption from d to 1 (as only encryptions of x are required). An additional advantage of this approach is that we only have to require the underlying encryption scheme to be IND-CPA secure (instead of satisfying pseudorandomness of ciphertexts).

HSS supporting SIMD operations. As first observed by [39], if the underlying ring R is of the right form, one can "pack" multiple plaintexts in one ciphertext. We show that our basic HSS supports "single instruction, multiple input" (SIMD) in this case. More precisely, we show that if $R = \mathbb{Z}[X]/(X^N + 1)$ for $N \in \mathbb{N}$, $N \leq \mathsf{poly}(\lambda)$ a power of 2, such that $X^N + 1$ splits over R_r (for some prime $r \geq 2$) into pairwise different irreducible polynomials of degree $k \in \mathbb{N}$ (i.e. $R_r \cong (\mathbb{F}_{r^k})^{N/k}$), one can evaluate a program P simultaneously on N/k inputs in $\mathbb{F}_{r^k}$. However, there are some caveats regarding magnitude growth with respect to the SIMD versus coefficient representations (see the full version for more details).

4 Instantiations and Efficiency Analysis

Our HSS schemes can be instantiated in a number of ways, using LWE or RLWE-based encryption schemes satisfying the nearly-linear decryption property from

$\underline{\mathsf{LPR.Gen}(1^\lambda):}$

1. Sample $a \leftarrow R_q, \widehat{s} \leftarrow \mathcal{D}_{\mathsf{sk}}, e \leftarrow \mathcal{D}_{\mathsf{err}}$ and compute $b = a \cdot \widehat{s} + e$ in R_q.
2. Let $\mathbf{s} = (1, \widehat{s})$ and output $\mathsf{pk} = (a, b), \mathsf{sk} = \mathbf{s}$.

$\underline{\mathsf{LPR.Enc}(\mathsf{pk}, m):}$

1. To encrypt $m \in R_p$, first sample $v \leftarrow \mathcal{D}_{\mathsf{sk}}, e_0, e_1 \leftarrow \mathcal{D}_{\mathsf{err}}$.
2. Output the ciphertext $(c_0, c_1) \in R_q^2$, where $c_1 = -av + e_0$ and $c_0 = bv + e_1 + (q/p) \cdot m$.

$\underline{\mathsf{LPR.OKDM}(\mathsf{pk}, m):}$

1. Compute $\mathbf{c}^0 = \mathsf{LPR.Enc}(0)$ and $\mathbf{c}^m = \mathsf{LPR.Enc}(m)$.
2. Output the tuple $(\mathbf{c}^m, \mathbf{c}^0 + (0, (q/p) \cdot m))$ as encryptions of $m \cdot \mathbf{s}$.

$\underline{\mathsf{LPR.DDec}(b, \mathbf{t}_b, \mathbf{c}^x):}$

1. Given $b \in \{0, 1\}$, a ciphertext $\mathbf{c}^x$ and a share $\mathbf{t}_b$ of $m \cdot \mathbf{s}$, first parse $\mathbf{c}^x = (c_0, c_1)$ and $\mathbf{t}_b = (t_{b,0}, t_{b,1})$.
2. Output $(d_0, d_1) := (\lfloor (p/q) \cdot (c_0 \cdot t_{b,0} + c_1 \cdot t_{b,1}) \rceil \mod p) \mod q$

Fig. 4: Ring-LWE based instantiation of PKE with approximately linear decryption, with procedures for HSS from Section 3

Definition 4. In this section we focus on a particularly efficient RLWE-based instantiation using a variant of the "LPR" encryption scheme [35] over 2-power cyclotomic rings. In the full version we also show how to use standard Regev encryption based on LWE [37], but this is less efficient in terms of share size.

4.1 Instantiation from Ring-LWE

Definition 5 (Decisional Ring Learning With Errors). *Let N be a power of 2, $q \geq 2$ be an integer, $R = \mathbb{Z}[X]/(X^N + 1)$ and $R_q = R/(qR)$. Let $\mathcal{D}_{\mathsf{err}}$ be an error distribution over R and $\mathcal{D}_{\mathsf{sk}}$ be a secret key distribution over R. Let $s \leftarrow \mathcal{D}_{\mathsf{sk}}$. The $\mathsf{RLWE}_{N,q,\mathcal{D}_{\mathsf{err}},\mathcal{D}_{\mathsf{sk}}}$ problem is to distinguish the following two distributions over R_q^2:*

- *$\mathcal{O}_{\mathcal{D}_{\mathsf{err}},s}$: Output (a, b) where $a \leftarrow R_q, e \leftarrow \mathcal{D}_{\mathsf{err}}$ and $b = a \cdot s + e$*
- *U: Output $(a, u) \leftarrow R_q^2$*

Formally, for a PPT adversary $\mathcal{A}$ we define the advantage $\mathrm{Adv}^{\mathsf{rlwe}}_{N,q,\mathcal{D}_{\mathsf{err}},\mathcal{D}_{\mathsf{sk}}}(\lambda) = |\Pr_{s \leftarrow \mathcal{D}_{\mathsf{sk}}}[\mathcal{A}^{\mathcal{O}_{\mathcal{D}_{\mathsf{err}},s}}(\lambda) = 1] - \Pr_{s \leftarrow \mathcal{D}_{\mathsf{sk}}}[\mathcal{A}^{U}(\lambda) = 1]|$.

In Figure 4 we present the core algorithms for our RLWE-based instantation using the LPR [35] public-key encryption scheme $\mathsf{LPR} = (\mathsf{LPR.Gen}, \mathsf{LPR.Enc})$, as

well as the auxiliary algorithms $\mathsf{LPR.OKDM}$ and $\mathsf{LPR.DDec}$ used by our HSS constructions. We use an error distribution $\mathcal{D}_{\mathsf{err}}$ where each coefficient is a rounded Gaussian with parameter σ, which gives $B_{\mathsf{err}} = 8\sigma$ as a high-probability bound on the ℓ_∞ norm of samples from $\mathcal{D}_{\mathsf{sk}}$, with failure probability $\mathtt{erf}(8/\sqrt{2}) \approx 2^{-49}$. We choose the secret-key distribution such that each coefficient of s is uniform in $\{0, \pm 1\}$, subject to the constraint that only h_{sk} coefficients are non-zero.[10]

The following lemma (proven in the full version) shows that LPR satisfies the nearly-linear decryption property for our HSS scheme. Furthermore, notice that ciphertexts output by $\mathsf{LPR.Enc}$ are pseudorandom under the decisional ring-LWE assumption, by a standard hybrid argument [36]. Therefore, the correctness and security properties of the $\mathsf{LPR.OKDM}$ and $\mathsf{LPR.DDec}$ procedures follow from Lemmas 3 and 4.

Lemma 6. *Assuming hardness of* $\mathsf{RLWE}_{N,q,\mathcal{D}_{\mathsf{err}},\mathcal{D}_{\mathsf{sk}}}$*, the scheme* LPR *(Figure 4) is a public-key encryption scheme with nearly-linear decryption over* $R = \mathbb{Z}[X]/(X^N + 1)$*, with ciphertext dimension* $d = 2$ *and bounds* B_{sk} *and* $B_{\mathsf{ct}} = B_{\mathsf{err}} \cdot (2h_{\mathsf{sk}} + 1)$*.*

4.2 Parameters and Efficiency Analysis

We now analyse the efficiency of our RLWE-based instantiation and compare it with using HSS constructed from somewhat homomorphic encryption, for various different settings of parameters.

For comparison with HSS based on DDH [7], we remark that for non-SIMD computations, DDH-based HSS shares can be smaller than both our approach and SHE. However, we estimate that homomorphic evaluation is around an order or magnitude faster than the times reported in [6] due to the expensive share conversion procedure, and when using SIMD both this and the share size can be dramatically improved.

Parameter estimation. We derived parameters for our HSS based on LPR using the bounds for correctness from Lemma 5, chosen to ensure that each RMS multiplication of a ring-element during evaluation is correct with probability $1-2^{-\kappa}$, where we chose $\kappa = 40$. To compare with constructing HSS from SHE, we estimated parameters for the "BFV" scheme based on RLWE [13, 24], currently one of the leading candidate SHE schemes. To modify this to achieve HSS with additive output sharing, we need to increase the size of q by around 2^κ bits. With both schemes we chose parameters estimated to have at least 80 bits of computational security, see the full version for more details.

Share size. Tables 1–2 show BFV ciphertext parameters for different multiplicative depths of circuit, and plaintext modulus 2 or $\approx 2^{128}$, respectively, to illustrate different kinds of Boolean and arithmetic computations. Table 3 gives

[10] Choosing a sparse secret like this does incur a small loss in security, and only gives us a small gain in parameters for the HSS. The main reason we choose s like this is to allow a fair comparison with SHE schemes, which typically have to use sparse secrets to obtain reasonable parameters.

our HSS parameters for various choices of B_{max}, the maximum value any plaintext coefficient can hold during the computation. Note that in contrast to SHE, our parameters depend only on this bound and not the multiplicative depth, although we are more restricted in that we can only perform homomorphic multiplications where one value is an input.

This means that comparing parameters of the two schemes is very application-dependent. For instance, for Boolean computations where we can have $B_{\mathsf{max}} = 2$, our scheme has smaller parameters than SHE for all computations of depth > 3, so this can give a significant advantage for very high degree functions that can be expressed as an RMS program. However, if SIMD computations are required then B_{max} must be chosen to account for the worst-case coefficient growth, which is not directly related to the plaintexts, so our scheme would likely have larger ciphertexts than SHE in most cases. For operations on large integers, the parameters in both schemes quickly get very large, though our parameters grow slightly quicker due to the increase in B_{max}.

Computational efficiency. The relative computational efficiency of the schemes is much clearer, and is the main advantage of our scheme over SHE. The cost of a homomorphic RMS multiplication with RLWE is roughly twice the cost of a decryption in any RLWE-based scheme (including BFV) with the same parameters. Recently, Halevi et al. [31] described an optimized implementation of BFV using CRT arithmetic, where according to their single-threaded runtimes, decryption costs between 20–30x less than multiplication (including key-switching) for the ranges of parameters we consider (cf. [31, Table 3]). This indicates a *10–15x improvement in performance* for homomorphic evaluation with our scheme compared with SHE, assuming similar parameters and numbers of multiplications. We remark that this comparison deserves some caution, since other SHE schemes such as BGV [14] may have different characteristics; we have not run experiments with BGV, but due to the complications in key-switching and modulus-switching we expect the improvement to still be around an order of magnitude.

5 Applications

In this section we highlight some applications of HSS for which our scheme seems well-suited. There are four primary approaches to compare: approaches not relying on HSS, using DDH-based or one-way function-based HSS, using HSS based on SHE, or using our new HSS. We remark that the concrete practicality of SHE-based HSS approaches has also not been considered before this work.

5.1 Secure 2-PC for low-degree polynomials

Perhaps the most natural application of HSS is to achieve a very succinct form of multi-party computation. After a setup phase to create the key material $\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)$, each party publishes HSS-shares of its input, which can then be directly used to compute additive shares of the output. Even the simplest case of evaluating degree-2 polynomials has many interesting applications, and also

Depth	N	$\log q$	Security
1	4096	102	145.1
2	4096	118	122.6
3	4096	134	106.2
4	4096	150	93.73
5	4096	164	85.53
6	8192	186	157.5
7	8192	202	142.9
8	8192	220	129.8
9	8192	236	120.1
10	8192	252	111.9

Table 1: BFV parameters with plaintext modulus 2

Depth	N	$\log q$	Security
1	16384	456	124.3
2	16384	602	92.44
3	32768	750	154.2

Table 2: BFV parameters with plaintext modulus $\approx 2^{128}$

B_{max}	N	$\log q$	Security
2	4096	137	103.3
2^{16}	4096	167	83.74
2^{32}	8192	203	142.0
2^{64}	8192	267	104.9
2^{128}	16384	399	143.9
2^{256}	16384	655	84.60

Table 3: RLWE based HSS parameters for RMS programs with maximum plaintext size B_{max}

allows us to use our optimized HSS scheme from Section 3.4, where shares consist of *a single* RLWE ciphertext, instead of two. The main motivating example we look at is to MPC protocols in the *preprocessing model*, where correlated randomness is pre-generated ahead of time to help increase efficiency when the actual computation takes place. This correlated randomness can take many forms, but the most common are Beaver triples, namely additive shares of (a, b, c) where $c = a \cdot b$ and a, b are random elements of a (typically) large prime field. These can easily be generated using degree-2 HSS, where each party inputs two field elements, and are also highly amenable to SIMD processing.

Looking at Tables 2–3, for an example of degree 2 functions over a 128-bit message space, BFV with depth 1 requires a dimension $N = 16384$ and modulus $\log q = 456$, whereas our scheme would need to use $B_{\mathsf{max}} \approx 2^{256}$, giving the same dimension and a slightly larger modulus of around 655 bits. Therefore, our communication cost will be slightly larger than using SHE-based HSS, but we expect to gain from the lower computational costs that come with our multiplication.

Using DDH-type HSS [6], an m-bit triple can be created with $3712(5m/4 + 160)$ bits of communication, giving 148kB for $m = 128$, meaning our communication is 20x higher for producing a single triple (at 2682kB), but orders of magnitude smaller ($\sim$900x) when amortized using SIMD (over $N = 16834$ triples). Computation requirements will greatly favor our approach.

We can also compare this with other approaches to Beaver triple generation. The SPDZ protocol [20] uses SHE (without HSS) to create triples; as well as the more complex homomorphic multiplication, this incurs extra costs in an interactive distributed decryption protocol, which adds a round of interaction that we can avoid using HSS with local rounding. The latest version of SPDZ [33] uses linearly-homomorphic encryption instead of SHE, and reports ciphertexts with $\log q$ as small as 327 bits, around half the size of ours. This would likely beat HSS in terms of communication and computation, but still has the undesirable

feature of 2 rounds of interaction, whereas with HSS (and a small one-time setup), the triples are obtained after just one message from each party.

The recent work of Boyle et al. [6] considered an interesting alternative approach to triple generation using so-called "cryptographic capsules", where HSS evaluation of a local PRG is used to expand a small, initial amount of correlated randomness into many more triples. This allows communication complexity *sublinear in the number triples*. They showed that this can be done with $O(\beta^2)$ Boolean RMS multiplications, where β is a parameter related to the locality of the PRG. With a DDH-like scheme, their protocol involves significant complications to ensure correct triples in the presence of a non-negligible failure probability for multiplication, making it quite impractical. However, using our HSS or SHE-based HSS with negligible error considerably simplifies this approach; it is not immediately clear of the best way to instantiate the parameters, but since it is a Boolean computation with relatively small degree it seems well-suited to our HSS scheme. We leave this exploration, as well as extending to other distributions, as an interesting direction of future research.

5.2 2-server PIR

An attractive application of HSS is to obtain highly succinct Private Information Retrieval (PIR) protocols for $m \geq 2$ servers. Here, m servers hold a public database DB and allow clients to submit private queries to DB, such that both the query and response remain hidden to up to $m-1$ colluding servers.[11] When using HSS, we can obtain a very simple, 1-round protocol where the client first sends an encryption of its query to both servers, who respond with an additive share of the result. Note that we only need the more efficient, secret-key version of HSS, such as our scheme from Section 3.4 with $m = 2$ servers.

Recent works on 2-server PIR have used HSS for point functions[12] to support basic queries including equality conditions, range queries and disjoint OR clauses, based on simple schemes using only one-way functions [9, 40]. However these techniques degrade dramatically for more complex queries, due to the relatively weak homomorphic ability of the underlying HSS. With HSS for branching programs we can significantly increase the expressiveness of queries, at the cost of some overhead in ciphertext size and running time.

In a bit more detail, suppose that a client issues a simple $COUNT$ query,[13] which applies some predicate Q to each row $x_i \in \mathsf{DB}$, and returns $\sum_i Q(x_i)$, that is, the number of rows in DB that match Q. The general idea is that the client splits Q into HSS shares s^1, s^2, and sends s^j to server j. For each row $x_i \in \mathsf{DB}$,

[11] Using S/FHE alone instead of HSS allows for the stronger setting of single-server PIR. However, a major advantage of HSS with additive reconstruction is that shares across many rows can easily be combined, allowing more expressive queries with simpler computation.

[12] Actually, these works use *function secret-sharing* [8] for point functions, which in this case is equivalent to HSS for the same class of functions.

[13] Other queries such as returning the record identifier, or min/max and range queries can easily be supported with similar techniques, as previously shown in [40, 6].

the servers then use homomorphic evaluation with the function $f_{x_i}(Q) := Q(x_i)$ on the shares, to obtain a shared 0/1 value indicating whether a match occurred. Given additive shares modulo r of the results $q_1, \ldots, q_D$ (where $D = |\mathsf{DB}|$), the servers can sum up the shares and send the result to the client, who reconstructs the result $q = \sum q_i$ (this assumes that $r < N$, so wraparound does not occur).

Below we analyse some useful classes of predicates that are much more expressive than function classes that can be handled using one-way function based approaches, and seem well-suited for our scheme supporting RMS programs.

Conjunctive keyword search. Suppose that each entry in DB is a document x with a list of keywords $W_x = \{w_1^x, \ldots, w_m^x\}$, and the query is a $COUNT$ query consisting of an arbitrary conjunction of keywords, each in $\{0,1\}^\ell$. That is, for a query $W = \{w_1, \ldots, w_k\}$ containing keywords shared bit-by-bit using the HSS, the servers will compute a sharing of

$$\#\{(x, W_x) \in \mathsf{DB} : W \subseteq W_x\}$$

To evaluate the query on a single entry of DB as an RMS program, we maintain the result f as a secret-shared memory value, which is initially set to 1. We then iterate over each query keyword $w_i \in W$, letting w_{ij} denote the j-th bit of w_i, and update f as

$$f := \sum_{w^x \in W_x} f \cdot \prod_{j=1}^{m} (1 \oplus w_j^x \oplus w_{ij})$$

Note that the i-th product evaluates to 1 iff $x^x = w_i$, and since all w^x are distinct, at most one of these will be 1. Multiplication by f applies a conjunction with the previous keyword, and must be performed inside the summation as f is a memory value. All other product terms are linear functions (over $\mathbb{Z}$) in the inputs w_i (via $a \oplus b = a + b - 2ab$), so each product can be evaluated left-to-right as an RMS program, for a total of $m \cdot \ell \cdot k$ RMS multiplications after iterating over all k query keywords.

Comparison to SHE-based HSS. When using SHE, the number of homomorphic multiplications is roughly the same as our case, and the multiplicative depth is $\log(m\ell k)$. For a concrete example, suppose that each document has $m = 10$ keywords of length $\ell = 128$ bits, and a client's query has $k = 4$ keywords. Using either our HSS scheme or HSS from SHE would need around 5120 multiplications per document, with a multiplicative depth of 13. This needs SHE parameters of $\log q \approx 300$ and dimension $N = 8192$ for the BFV scheme as above, whereas with our scheme we can use the best case of $B_{\mathsf{max}} = 2$, giving $\log q \approx 137$ and $N = 4096$. Using our secret-key HSS and LPR instantiation, the share size is $3N \log q$ bits $\approx$ 210kB, around 1/3 of the SHE ciphertext size using BFV. The communication cost for the whole query would be 107MB for our HSS, and 314MB with BFV, whilst we estimate the computational costs of homomorphic evaluation per document are around 2.5s and 300s, respectively, so even with the relatively high communication cost, for matching several documents using our HSS would certainly give a significant performance improvement.

However, one drawback of our approach is that handling SIMD computations is more challenging, since the B_{max} bound must be chosen much larger to account for the coefficient growth of the plaintext polynomials, which may continue to grow even when the packed plaintext messsages themselves are only bits. If the number of documents in the database is large enough to warrant SIMD processing then it seems likely that SHE will be preferable, since $N = 8192$ documents could be searched at once without increasing the parameters.

Pattern-matching queries. Suppose here that the client wants to search for the occurrence of a pattern $p = (p_1, \ldots, p_m) \in \{0,1\}^m$ in each row $x = (x_1, \ldots, x_N) \in \{0,1\}^N$. An RMS program for computing the pattern-matching predicate, with public input x and private input p, can be done with $m \cdot N$ multiplications using a similar method to the previous example, modified slightly to compute the OR of matching p with every position in x.

Comparison to SHE-based HSS. When using SHE, this computation has depth $\log(nm)$, also requiring around $N \cdot m$ homomorphic multiplications. The comparison with our scheme is then similar to the keyword search example, depending on the parameters chosen. For another example, if we have a fairly large string of length $N = 10000$, and a pattern of size $m = 100$, then the SHE-based HSS must support depth 20, giving parameters $(N, \log q) = (16384, 434)$. Again, we can use our HSS with parameters for $B_{\mathsf{max}} = 2$, which lead to ciphertexts around 8.5x smaller than with SHE.

Acknowledgements

We would like to thank the anonymous reviewers of Eurocrypt 2019 for their thorough and generous comments.

References

[1] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. "Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems". In: *CRYPTO 2009*. LNCS. Springer, Heidelberg, Aug. 2009.

[2] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. "Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE". In: *EUROCRYPT 2012*. LNCS. Springer, Heidelberg, Apr. 2012.

[3] Abhishek Banerjee, Chris Peikert, and Alon Rosen. "Pseudorandom Functions and Lattices". In: *EUROCRYPT 2012*. LNCS. Springer, Heidelberg, Apr. 2012.

[4] Florian Böhl, Gareth T. Davies, and Dennis Hofheinz. "Encryption Schemes Secure under Related-Key and Key-Dependent Message Attacks". In: *PKC 2014*. LNCS. Springer, Heidelberg, Mar. 2014.

[5] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. "Compressing Vector OLE". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*. 2018.

[6] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. "Homomorphic Secret Sharing: Optimizations and Applications". In: *ACM CCS 2017*. ACM Press, 2017.

[7] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Breaking the Circuit Size Barrier for Secure Computation Under DDH". In: *CRYPTO 2016, Part I*. LNCS. Springer, Heidelberg, Aug. 2016.

[8] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function Secret Sharing". In: *EUROCRYPT 2015, Part II*. LNCS. Springer, Heidelberg, Apr. 2015.

[9] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function Secret Sharing: Improvements and Extensions". In: *ACM CCS 2016*. ACM Press, Oct. 2016.

[10] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Group-Based Secure Computation: Optimizing Rounds, Communication, and Computation". In: *EUROCRYPT 2017, Part II*. LNCS. Springer, Heidelberg, 2017.

[11] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. "Foundations of Homomorphic Secret Sharing". In: *ITCS 2018*. LIPIcs, Jan. 2018.

[12] Elette Boyle, Lisa Kohl, and Peter Scholl. *Homomorphic Secret Sharing from Lattices Without FHE*. Cryptology ePrint Archive, Report 2019/129. `https://eprint.iacr.org/2019/129`.

[13] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP". In: *CRYPTO 2012*. LNCS. Springer, Heidelberg, Aug. 2012.

[14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping". In: *ITCS 2012*. ACM, Jan. 2012.

[15] Zvika Brakerski and Vinod Vaikuntanathan. "Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages". In: *CRYPTO 2011*. LNCS. Springer, Heidelberg, Aug. 2011.

[16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds". In: *ASIACRYPT 2016, Part I*. LNCS. Springer, Heidelberg, Dec. 2016.

[17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE". In: *ASIACRYPT 2017, Part I*. LNCS. Springer, Heidelberg, Dec. 2017.

[18] Richard Cleve. "Towards Optimal Simulations of Formulas by Bounded-Width Programs". In: *Computational Complexity* 1 (1991), pp. 91–105. DOI: `10.1007/BF01200059`. URL: `https://doi.org/10.1007/BF01200059`.

[19] 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*. IEEE Computer Society Press, May 2015.

[20] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *CRYPTO 2012*. LNCS. Springer, Heidelberg, Aug. 2012.

[21] Itai Dinur, Nathan Keller, and Ohad Klein. "An Optimal Distributed Discrete Log Protocol with Applications to Homomorphic Secret Sharing". In: *CRYPTO 2018, Part III*. LNCS. Springer, Heidelberg, Aug. 2018.

[22] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. "Spooky Encryption and Its Applications". In: *CRYPTO 2016, Part III*. LNCS. Springer, Heidelberg, Aug. 2016.

[23] Léo Ducas and Daniele Micciancio. "FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second". In: *EUROCRYPT 2015, Part I*. LNCS. Springer, Heidelberg, Apr. 2015.

[24] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2012/144. `http://eprint.iacr.org/2012/144`. 2012.

[25] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith III. "Homomorphic Secret Sharing from Paillier Encryption". In: *Provable Security - 11th International Conference, ProvSec 2017, Proceedings*. 2017, pp. 381–399.

[26] Craig Gentry. "Fully homomorphic encryption using ideal lattices". In: *41st ACM STOC*. ACM Press, 2009.

[27] Craig Gentry, Shai Halevi, and Nigel P. Smart. "Homomorphic Evaluation of the AES Circuit". In: *CRYPTO 2012*. LNCS. Springer, Heidelberg, Aug. 2012.

[28] Craig Gentry, Amit Sahai, and Brent Waters. "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based". In: *CRYPTO 2013, Part I*. LNCS. Springer, Heidelberg, Aug. 2013.

[29] Niv Gilboa and Yuval Ishai. "Distributed Point Functions and Their Applications". In: *EUROCRYPT 2014*. LNCS. Springer, Heidelberg, May 2014.

[30] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *19th ACM STOC*. ACM Press, May 1987.

[31] Shai Halevi, Yuriy Polyakov, and Victor Shoup. *An Improved RNS Variant of the BFV Homomorphic Encryption Scheme*. Cryptology ePrint Archive, Report 2018/117. `https://eprint.iacr.org/2018/117`. 2018.

[32] Shai Halevi and Victor Shoup. "Bootstrapping for HElib". In: *EUROCRYPT 2015, Part I*. LNCS. Springer, Heidelberg, Apr. 2015.

[33] Marcel Keller, Valerio Pastro, and Dragos Rotaru. "Overdrive: Making SPDZ Great Again". In: *EUROCRYPT 2018, Part III*. LNCS. Springer, Heidelberg, 2018.

[34] Adeline Langlois and Damien Stehlé. "Worst-case to average-case reductions for module lattices". In: *Des. Codes Cryptography* 75.3 (2015), pp. 565–599.

[35] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "A Toolkit for Ring-LWE Cryptography". In: *EUROCRYPT 2013*. LNCS. Springer, Heidelberg, May 2013.

[36] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *EUROCRYPT 2010*. LNCS. Springer, Heidelberg, 2010.

[37] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *37th ACM STOC*. ACM Press, May 2005.

[38] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. "On data banks and privacy homomorphisms". In: *Foundations of secure computation (Workshop, Georgia Inst. Tech., 1977)*. Academic, New York, 1978, pp. 169–179.

[39] N. P. Smart and F. Vercauteren. "Fully Homomorphic SIMD Operations". In: *Des. Codes Cryptography* 71.1 (Apr. 2014), pp. 57–81. ISSN: 0925-1022. DOI: `10.1007/s10623-012-9720-4`.

[40] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. "Splinter: Practical Private Queries on Public Data". In: *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017*. 2017, pp. 299–313.