On Homomorphic Secret Sharing from Polynomial-Modulus LWE

Thomas Attema[1,2,3], Pedro Capitão[1,2], and Lisa Kohl[1]

[1] CWI, Cryptology Group, Amsterdam, The Netherlands
{pedro,lisa.kohl}@cwi.nl
[2] Leiden University, Mathematical Institute, Leiden, The Netherlands
[3] TNO, Cyber Security and Robustness, The Hague, The Netherlands
thomas.attema@tno.nl

Abstract. Homomorphic secret sharing (HSS) is a form of secret sharing that supports the local evaluation of functions on the shares, with applications to multi-server private information retrieval, secure computation, and more.

Insisting on additive reconstruction, all known instantiations of HSS from "Learning with Error (LWE)"-type assumptions either have to rely on LWE with superpolynomial modulus, come with non-negligible error probability, and/or have to perform expensive ciphertext multiplications, resulting in bad concrete efficiency.

In this work, we present a new 2-party local share conversion procedure, which allows to *locally* convert noise encoded shares to non-noise plaintext shares such that the parties can detect whenever a (potential) error occurs and in that case resort to an alternative conversion procedure.

Building on this technique, we present the first HSS for branching programs from (Ring-)LWE with polynomial input share size which can make use of the efficient multiplication procedure of Boyle et al. (Eurocrypt 2019) and has no correctness error. Our construction comes at the cost of a – on expectation – slightly increased output share size (which is insignificant compared to the input share size) and a more involved reconstruction procedure.

More concretely, we show that in the setting of 2-server private information retrieval we can choose ciphertext sizes of only a quarter of the size of the scheme of Boyle et al. at essentially no extra cost.

1 Introduction

In 1979, Shamir introduced the concept of secret sharing information in his seminal paper *How to Share a Secret* [31]. In the two-party setting, secret sharing allows to split up a secret value into two secret shares, such that each share individually hides the secret, whereas the shares together allow to recover it. The simplest secret-sharing scheme is *additive secret sharing*, where a value x in an additive group $\mathbb{G}$ is split into x_0, x_1, such that x_0, x_1 are distributed uniformly at random conditioned on $x_0 + x_1 = x$. Despite its simplicity, additive secret sharing comes with a

number of nice properties. For example, it allows the *local* evaluation of linear functions on the shares.

In 2019, Boyle, Gilboa and Ishai [10] extended this notion to *homomorphic secret sharing (HSS)*, which allows the *local* evaluation of larger classes of function on the shares, while keeping the nice properties of additive secret sharing (so far possible). More precisely, a homomorphic secret-sharing scheme for a function class $\mathcal{F}$ (over some input space $\mathbb{G}$) has the following properties:

- The secret shares individually hide the message (*computationally*).
- The secret shares are succinct, i.e., they are *polyomial* in the size of the secret to be shared (in particular, they are independent of the complexity of the function class $\mathcal{F}$).
- The secret shares allow local evaluation of all functions $f \in \mathcal{F}$. More precisely, there exists an evaluation procedure Eval, such that given secret shares x_0, x_1 of $x \in \mathbb{G}$, it holds $\mathsf{Eval}(f, x_0) + \mathsf{Eval}(f, x_1) = f(x)$.

Note that the last condition explicitly requires *additive reconstruction*, i.e., evaluation results in an additive secret sharing of the output. While this requirement can be relaxed to more general reconstruction functions (as we will do in this work), it has a number of useful features, such as allowing the local postprocessing with linear functions.

Since their introduction, homomorphic secret sharing has found numerous applications, including 2-server private-information retrieval [24, 9, 19, 11, 32], low-communication secure computation [10, 12, 8, 20], and succinct generation of correlated (pseudo-)randomness [6, 7].

In [10], Boyle et al. presented a homomorphic secret-sharing scheme from the decisional Diffie-Hellman assumption for the class of *restricted multiplication straight-line (RMS) programs*. These programs are restricted in that they only allow multiplication between an input value and a memory value (where a memory value is an intermediate value in the computation), but not a multiplication between two memory values. It can be shown that this captures the class of polynomial-size branching programs, and circuits of constant fan-out and logarithmic depth (i.e., circuits in the complexity class NC^1).

Since then, further HSS constructions for RMS programs have been proposed based on the decisional Diffie-Hellman assumption [8], the Paillier assumption [23, 28, 30], and based on the learning with errors (LWE) assumption [22, 14, 16]. All schemes, however, come with some efficiency bottleneck: either the evaluation is computationally expensive [10, 22, 23, 8, 28, 30, 16] and/or the input shares have high concrete overhead resulting in bad communication complexity [22, 14, 16].

In particular, while the scheme of Boyle et al. BKS [14] comes with desirable properties such as (plausible) post-quantum security and (comparatively) efficient multiplication on ciphertexts, it inherently has to rely on LWE with (double-)superpolynomial modulus (and thus large ciphertexts) in order to keep the error probability negligible. The reason for their (double-)superpolynomial modulus is a share conversion procedure to locally convert noise encoded shares modulo q to non-noisy shares modulo q. In order to achieve negligible error probability, they need to choose moduli p, q with $1 \ll p \ll q$, where each $\ll$ denotes a superpolynomial gap. The starting point for our work can thus be phrased as follows.

Is it possible to design a share-conversion procedure for polynomial-sized p, q without introducing a non-negligible error?

1.1 Our Contribution

In this paper, we answer this question (somewhat) affirmatively and present an HSS scheme from LWE for RMS programs with polynomial modulus, which otherwise inherits the nice properties from BKS. Our core technique is a share conversion which allows to locally detect and tentatively correct potential errors. On the downside, we have to relax additive reconstruction to a more involved reconstruction procedure, where the parties choose the output from an expected constant-size list of potential output values. In the following we give a high-level overview of our main results, which we discuss in more detail in the technical overview.

Our core lemmas. Our core technique can be captured in the following two lemmas for *share conversion*, a crucial step in the homomorphic evaluation of multiplications. Informally, the lemma states that (for rounding) there exist *local* conversion procedures that return shares flag_0, z_0 and $\mathsf{flag}_1, z_1, z_1'$, respectively, such that either $z_0 = z_1 \mod p$ or $z_0 = z_1' \mod p$, where the latter holds if and only if $\mathsf{flag}_0 = \mathsf{flag}_1 = 1$. This extends the technique of BKS, who only consider the case $\mathsf{flag}_0 = \mathsf{flag}_1 = 0$ and choose parameters to ensure that this holds except with negligible probability.

Lemma 1 (Rounding with correction [Lemma 5, 6]). *Let $p, q \in \mathbb{N}$ with $p|q$. Then, there exist efficient procedures $\mathsf{Round}_0 \colon \mathbb{Z}_q \to \{0,1\} \times \mathbb{Z}_p$ and $\mathsf{Round}_1 \colon \mathbb{Z}_q \to \{0,1\} \times \mathbb{Z}_p^2$ such that the following holds:*
For any $x \in \mathbb{Z}_p$, any $e \in \mathbb{Z}$ with $|e| < q/(4p)$, and any t_0, t_1 with

$$t_0 + t_1 = \frac{q}{p} \cdot x + e \mod q,$$

it holds

$$x = \begin{cases} z_0 + z_1 \mod p & \text{if } \mathsf{flag}_0 = 0 \vee \mathsf{flag}_1 = 0\,, \\ z_0 + z_1' \mod p & \text{if } \mathsf{flag}_0 = \mathsf{flag}_1 = 1\,, \end{cases}$$

where $(\mathsf{flag}_0, z_0) \leftarrow \mathsf{Round}_0(t_0)$ and $(\mathsf{flag}_1, z_1, z_1') \leftarrow \mathsf{Round}_1(t_1)$.
Further, for t_0, t_1 chosen at random, it holds $\mathsf{flag}_0 = \mathsf{flag}_1 = 0$ with probability at least $1 - (4 \cdot |e| \cdot p)/q$.

Similarly, we extend their lemma for lifting.

Lemma 2 (Lifting with correction [Lemma 8, 9]). *Let $p, q \in \mathbb{N}$ with $p|q$. Then, there exist efficient procedures $\mathsf{Lift}_0 \colon \mathbb{Z}_p \to \{0,1\} \times \mathbb{Z}_q$ and $\mathsf{Lift}_1 \colon \mathbb{Z}_p \to \{0,1\} \times \mathbb{Z}_q^2$ such that the following holds:*
For any $x \in \mathbb{Z}_p$, with $|x| < p/6$, and any z_0, z_1 with

$$z_0 + z_1 = x \mod p,$$

it holds

$$x = \begin{cases} v_0 + v_1 \mod q & \text{if } \mathsf{flag}_0 = 0 \vee \mathsf{flag}_1 = 0\,, \\ v_0 + v_1' \mod q & \text{if } \mathsf{flag}_0 = \mathsf{flag}_1 = 1\,, \end{cases}$$

where $(\mathsf{flag}_0, v_0) \leftarrow \mathsf{Lift}_0(z_0)$ *and* $(\mathsf{flag}_1, v_1, v_1') \leftarrow \mathsf{Lift}_1(z_1)$.
Further, for z_0, z_1 *chosen at random it holds* $\mathsf{flag}_0 = \mathsf{flag}_1 = 0$ *with probability at least* $1 - (4 \cdot |x|)/p$.

Our HSS. We show that building on the core lemma, we obtain an HSS with one-sided error correction. More precisely, $\mathcal{P}_0$ will follow a fixed computation path (remembering the wires where $\mathsf{flag}_0 = 1$). Party $\mathcal{P}_1$ on the other hand, continues the computation for z_1 and z_1' whenever $\mathsf{flag}_1 = 1$ for some wire. In the end, the parties can reconstruct the value by choosing the computation path that resorts to the alternative computation for $\mathcal{P}_1$ whenever $\mathsf{flag}_0 = 1$ and $\mathsf{flag}_1 = 1$ for some wire. Note that this potentially results in *exponential* computation time for $\mathcal{P}_1$. We resolve this by choosing the parameters depending on the number of multiplications to be performed, such that the *overall* number of expected errors is 1 (or less). This means that on expectation $\mathcal{P}_1$ has to perform the computation twice (from some point in the program on) and finally obtains two output shares. We want to stress that the output shares (corresponding to plaintext values) are typically several orders of magnitude smaller than the input shares (corresponding to ciphertext values). The increase in output values is therefore insignificant compared to the savings in input shares.
For instantiating our HSS, we present a trade-off between ciphertext size (equaling the input share size) and expected number of output shares. More precisely, instantiating the underlying public-key encryption scheme PKE with the Ring-LWE based encryption scheme of Lyubashevsky, Peikert and Regev [27] over the ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, we obtain the following.

Lemma 3 (Corollary of Lemma 11). *Let* $\gamma > 1$. *Let* P *be a branching program with multiplicative size* $|P|$ *(i.e., number of load and multiplication operations) and magnitude bound* B_{max} *(i.e., upper bound on all intermediary computation values). Then, setting* $p \geq 8 \cdot B_{\mathsf{max}} \cdot N \cdot |P| / \ln \gamma$ *and* $q \geq 8 \cdot p \cdot N \cdot |P| / \ln \gamma$ *in our HSS construction party* $\mathcal{P}_1$ *obtains at most* γ *output shares on expectation.*

Setting $\gamma = 1 + \lambda^{-\omega(\log \lambda)}$ (and thus obtaining $1/\ln \gamma \approx \lambda^{\omega(\log \lambda)}$) we can recover the negligible error probability at the cost of superpolynomial ciphertext sizes of BKS.

HSS with perfect correctness. As a corollary of our techniques, we can obtain an HSS for RMS programs that satisfies *perfect* correctness, since the parties can *always* detect and correct the errors.

B_{max}	N	$\log q$
2	2048	71
2^{16}	2048	86
2^{32}	4096	104
2^{64}	4096	136
2^{128}	8192	202
2^{256}	8192	330

Table 1: Our HSS parameters for program size $|P| = 2^{20}$, $\gamma = 2$.

B_{max}	N	$\log q$
2	4096	137
2^{16}	4096	167
2^{32}	8192	203
2^{64}	8192	267
2^{128}	16384	399
2^{256}	16384	655

Table 2: BKS HSS parameters with *per gate* error probability 2^{-40}.

Concrete efficiency. In Tables 1 and 2, we give concrete parameter sizes in comparison with the scheme of BKS, depending on the program size $|P|$. Note that the parameters of the BKS HSS scheme also have to grow with the program size of the underlying program $|P|$ to ensure a fixed error probability, similarly to our scheme. Even without taking this into account (i.e., considering an error probability of 2^{-40} after one operation rather than $|P|$), it can be seen that our scheme can achieve a factor 4 shorter ciphertexts.

HSS with expected constant-time evaluation. The focus of our paper are applications where there is no privacy requirement for reconstruction, and thus *expected* constant-time evaluation can be dealt with by cutting off the computation after a fixed certain number of operations. We note though that the expected running time of the evaluation algorithms imposes challenges in applications such as secure two-party computation, where party $\mathcal{P}_0$ can potentially derive information about the input from the response time of $\mathcal{P}_1$. We leave dealing with this issue as an interesting open question.

Share reconstruction with privacy. We note that (apart from the above described problem concerning run-time leakage) the problem of share reconstruction with privacy can be viewed as (one-server) private information retrieval by keywords [17] satisfying a strong notion of database privacy, where the client (here party $\mathcal{P}_0$) is not allowed to learn anything about the number and content of the database held by the server (here party $\mathcal{P}_1$), except for the queried entry. This can be viewed as a special case of labelled private-set intersection [15, 18] and can be instantiated by relying on somewhat homomorphic encryption. (Note here that the database for share reconstruction is very small on expectation, and thus even using expensive ciphertext multiplication for the final reconstruction would in typical applications not have a significant impact on the overall run time.)

Impossibility of fully local share conversion. To complement our result, we show that *no* direct local share conversion (i.e., not resorting to

an alternative conversion procedure) can achieve negligible error, showing that the BKS HSS scheme inherently requires either superpolynomial ciphertext or some postprocessing on the outputs.

Limitation to 2-party HSS. As for BKS, our techniques are inherently limited to the two-party case, since we use some "symmetry" properties between the two shares. More precisely, we rely on the fact that if $t_0+t_1 = \frac{q}{p} \cdot x + e$, then the distance of t_0 and t_1 to the next (potentially different) multiple of $\frac{q}{p}$ differs only by $|e|$. This is no longer true for three or more parties, where local rounding results in a constant error probability (independent of p and q). Going beyond the two-party case therefore inherently requires new techniques.

Beyond HSS. A corollary of our core lemma is that the secure reconstruction of $x \mod p$ given $t_0 + t_1 = \frac{q}{p} \cdot x + e$ can be performed using a single string-OT, where party $\mathcal{P}_0$ acts as the sender with input-bit flag_0 and $\mathcal{P}_1$ acts as the receiver inputting (z_1, z_1) if $\mathsf{flag}_1 = 0$ and (z_1, z_1') else. This might have applications to encryption with 2-party distributed decryption, as used, e.g., in lattice-based electronic voting schemes.

HSS rounding vs. learning with rounding (LWR). The rounding function which underlies [14] and this paper is essentially the same as the rounding function used for LWR [4]. While [4] uses non-distributed rounding to reduce the hardness of LWR to LWE (essentially building on the fact that the LWE error is "rounded away" with high probability), the line of work on constructing HSS via rounding needs a stronger property on distributed rounding towards achieving correctness. In particular, the techniques to reduce the modulus in the reduction from LWR to LWE from super-polynomial to polynomial [2, 5] do not appear to help in reducing the modulus for LWE-based HSS constructions.

1.2 Technical Overview

In the following, we give an overview of the idea behind our core lemma and our HSS construction. For the purpose of the technical overview, we assume $\mathcal{R} = \mathbb{Z}$, $n \in \mathbb{N}$, and $p = p(\lambda), q = q(\lambda) \in \mathbb{N}$ such that $p|q$. By writing $p \ll q$, we denote that $q/p \in \lambda^{\omega(1)}$.

Restricted Multiplication Straight-Line Programs (RMS). Recall that for RMS programs there is a distinction between *input values* (inputs to the program) and *memory values* (intermediary computation values) and the following operations are supported:

- Loading an input value into memory;
- Adding two memory values;
- Multiplying an input value with a memory value;
- Outputting a memory value.

The HSS scheme of [14]. Our starting point is the HSS scheme of [14]. The basis of their construction is an encryption scheme with nearly linear encryption. More precisely, let $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme over message space $\mathbb{Z}_p$, such that the secret key and ciphertext space is $\mathbb{Z}_q^d$. Recall that PKE satisfies *nearly linear decryption*, if for all secret keys $\mathbf{s}$, all messages $m \in \mathbb{Z}_p$, and and all encryptions $\mathbf{c}$ of $\mathbf{m}$, it holds

$$\langle \mathbf{s}, \mathbf{c} \rangle \approx \frac{q}{p} \cdot m \mod q.$$

Further, BKS requires that $\mathbf{s}$ has only entries in $\{-1, 0, 1\}$ (or otherwise small bounded values). As observed in [14], these requirements are indeed satisfied by (variants of) many lattice-based encryption schemes [29, 3, 26, 4, 25].

Now, if $B_{\mathsf{max}} \in \mathbb{N}$ with $B_{\mathsf{max}} \ll p \ll q/B_{\mathsf{max}}$, then an HSS for RMS programs with magnitude bound B_{max} can be obtained as follows.

Key generation. The HSS key generation generates a key pair according to the key generation algorithm $\mathsf{PKE.Enc}$ and outputs secret key shares $\mathsf{ek}_0 := \mathbf{s}_0$ to $\mathcal{P}_0$ and $\mathsf{ek}_1 := \mathbf{s}_1$ to $\mathcal{P}_1$, s.t., $\mathbf{s}_0 + \mathbf{s}_1 = \mathbf{s}$ for the secret key $\mathbf{s} \in \{0, 1\}^d$.

Input and memory values. Values are stored as follows.

- **Input values:** Input values $|x| \leq B$ are encrypted as $\{\mathsf{Enc}(x \cdot s_i)\}_{i \in [d]}$, where s_i is the i-th component of $\mathbf{s}$. (Note that by the techniques of BKS this is possible given knowledge only of the public key of the underlying encryption scheme. We will give more details on this in the main body of the paper.)
- **Memory values:** Memory values $|y| \leq B$ are secret shared as $\mathbf{t}_0, \mathbf{t}_1$, such that $\mathbf{t}_0 + \mathbf{t}_1 = y \cdot \mathbf{s} \mod q$.

Note that adding two memory values is straightforward by the linearity of additive secret sharing. Further, assuming that the first component of the secret key $\mathbf{s}$ is always one (which is straightforward to achieve), outputting a memory value mod q can be done by simply outputting the first entry of the corresponding share. Finally, loading an input value is equivalent to multiplying an input value by 1. We therefore restrict to describing the restricted multiplication in the following.

To perform a multiplication of an input value x encrypted as $\{\mathbf{c}_i\}_{i \in [d]}$ with a memory value y shared as $(\mathbf{t}_0, \mathbf{t}_1)$, the idea is for the parties to locally compute $\mathbf{t}_b^{\mathsf{pre}}$ as $t_{b,i}^{\mathsf{pre}} := \langle \mathbf{c}_i, \mathbf{t}_b \rangle$. By the property of nearly linear decryption, this yields:

$$t_{0,i}^{\mathsf{pre}} + t_{1,i}^{\mathsf{pre}} = \langle \mathbf{c}_i, y \cdot \mathbf{s} \rangle = y \cdot \langle \mathbf{c}_i, \mathbf{s} \rangle \approx \frac{q}{p} \cdot x \cdot y \cdot s_i \mod q,$$

and thus

$$\mathbf{t}_0^{\mathsf{pre}} + \mathbf{t}_1^{\mathsf{pre}} \approx \frac{q}{p} \cdot x \cdot y \cdot \mathbf{s} \mod q.$$

The challenging part is to *locally* convert the shares $\mathbf{t}_b^{\mathsf{pre}}$ into memory values, i.e., $\mathbf{t}_0^{\mathsf{out}} + \mathbf{t}_1^{\mathsf{out}} = x \cdot y \cdot \mathbf{s} \mod q$. To that end, BKS [14] introduce the *rounding* and *lifting* technique, which allow local share conversion. In the following, we will focus on the *rounding* technique, since the *lifting* technique (to lift shares modulo p to shares modulo q) can be adapted similarly.

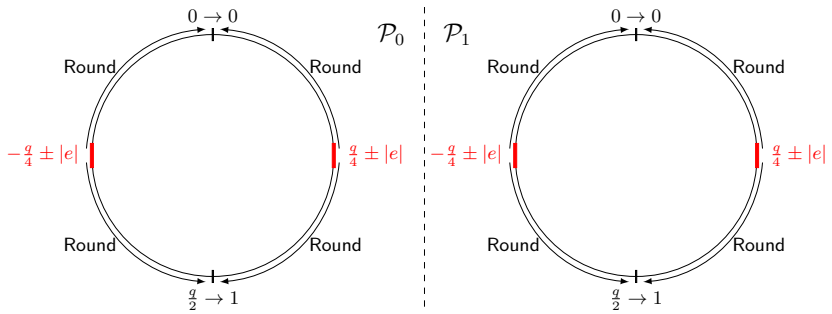

Fig. 1: Depiction of the local rounding procedure. If both shares are *outside* the area highlighted in red, then no rounding error occurs.

Lemma 4 (Rounding [BKS [14]]). *Let $p, q \in \mathbb{N}$ such that $p|q$. Let $x \in \mathbb{Z}_p$ and let $e \in \mathbb{Z}$ with $|e| \ll q/p$. Let $t_0, t_1 \in \mathbb{Z}_q$ be sampled uniformly at random subject to*

$$t_0 + t_1 = \frac{q}{p} \cdot x + e \mod q.$$

Then there exists an efficient deterministic procedure Round *such that*

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

except with negligible probability.

Towards HSS from polynomial-modulus LWE. A straightforward approach towards HSS with polynomial modulus is to choose p, q of polynomial-size and handle the resulting non-negligible error with the generic error correction techniques of [10] introduced towards HSS from decisional Diffie-Hellman (where a non-negligible error is inherent [21]). These generic error correcting techniques come with a high concrete overhead though: If the error probability is a constant, then $\omega(\log \lambda)$-repetitions are necessary to achieve negligible error-probability via a majority vote. Thus, both the evaluation time and the size of the output shares are increased by a factor of $\omega(\log \lambda)$.

This work: HSS from polynomial-modulus LWE with fine-grained error correction. In this work, we show that in the case of LWE – and unlike decisional Diffie-Hellman – it is actually possible to detect (potential) errors, and therefore only correct if an error really occurs (or is very likely to occur). In order to outline our techniques, in the following we take a closer look at the rounding procedure from above.
To simplify presentation, for the rounding technique we assume $p = 2$ and $4|q$ (to ensure $\frac{q}{2}$ and $\frac{q}{4}$ are integers). We give a depiction of the rounding procedure in Figure 1, where $\mathsf{Round}\colon \mathbb{Z}_q \to \mathbb{Z}_2$ is defined as

$$\mathsf{Round}(y) := \left\lfloor \frac{2}{q} \cdot y \right\rceil \mod 2.$$

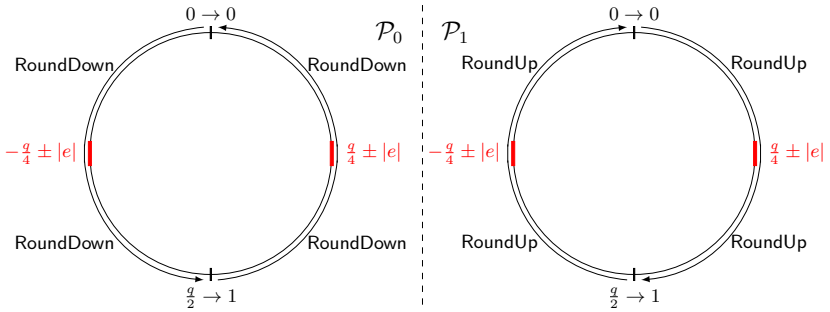

Fig. 2: Depiction of the alternative local rounding procedure. If at least one of the shares is *inside* the area highlighted in red, then no rounding error occurs.

Now, assume to be given shares t_0, t_1 chosen at random conditioned on

$$t_0 + t_1 = \frac{q}{2} \cdot x + e,$$

where $x \in \{0, 1\}$ and e is some error. Then, as observed in BKS [14], if at least one of the shares t_0, t_1 is *outside* the *red area* $\left[-\frac{q}{4} \pm |e|\right] \cup \left[\frac{q}{4} \pm |e|\right]$,[4] then no rounding error occurs, i.e.,

$$\left\lfloor \frac{2}{q} \cdot t_0 \right\rceil + \left\lfloor \frac{2}{q} \cdot t_1 \right\rceil = x \mod 2.$$

This crucially relies on the fact that for the shares it holds that $t_0 + t_1 = e \mod q$ or $t_0 + t_1 = \frac{q}{2} + e \mod q$. Now, assume t_0 is outside the red area and $\mathsf{Round}(t_0) = 0$ (the other cases are similar). Then, it must hold that t_0 has distance $< \frac{q}{4} - |e|$ from 0. Thus, if $t_0 + t_1 = e$, it must hold that t_1 has distance $< \frac{q}{4}$ from 0, and thus $\mathsf{Round}(t_1) = 0$ as required. On the other hand, if $t_0 + t_1 = \frac{q}{2} + e \mod q$, then t_1 must have distance $< \frac{q}{4}$ from $\frac{q}{2}$, and thus $\mathsf{Round}(t_1) = 1$ as required.

If $|e| \ll \frac{q}{2}$, then the probability of a random element $y \xleftarrow{\$} \mathbb{Z}_q$ lying in the red area is negligible, and thus by the above considerations no rounding error occurs except with negligible probability.

Towards correcting the error, we observe that – on the other hand – if at least one of the shares t_0, t_1 is *inside* one of the bad areas, then following an alternative procedure (depicted in Figure 2) no rounding error occurs. The alternative rounding procedures $\mathsf{RoundDown}, \mathsf{RoundUp}$ are defined as

$$\mathsf{RoundDown}(x) := \left\lfloor \frac{2}{q} \cdot x \right\rfloor \mod 2, \quad \mathsf{RoundUp}(x) := \left\lceil \frac{2}{q} \cdot x \right\rceil \mod 2.$$

[4] Here, we consider $\mathbb{Z}_q$ to be represented as integers in the interval $\left(-\frac{q}{2}, \frac{q}{2}\right]$. For $y \in \left\{-\frac{q}{4}, \frac{q}{4}\right\}$, by $[y \pm |e|]$ we denote the interval containing all $z \in \mathbb{Z}_q$ having at most distance $|e|$ from y (considered as integer).

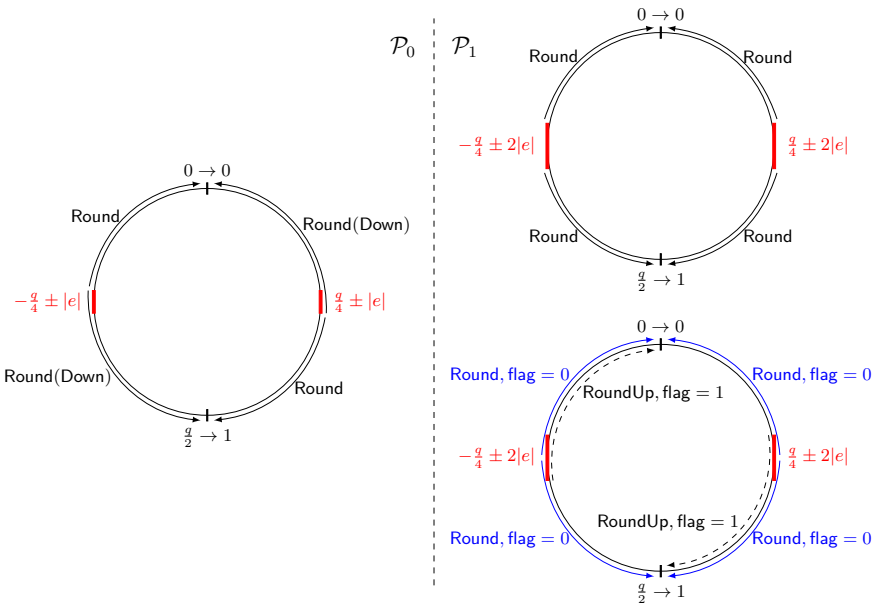

Fig. 3: Depiction of the asymmetric local rounding procedure, where party $\mathcal{P}_1$ is fully in charge of the error correction.

In other words, party $\mathcal{P}_0$ rounds all negative numbers $\left(-\frac{q}{2}, -1\right]$ to $-1 = 1 \mod 2$, and all positive number $\left[1, \frac{q}{2}\right]$ to 0, and $\mathcal{P}_1$ rounds all negative numbers $\left(-\frac{q}{2}, -1\right]$ to 0, and all positive numbers $\left[1, \frac{q}{2}\right]$ to 1 (and 0 is always rounded to 0).

The idea here is that if at least one of the shares t_0, t_1 is *inside* the red area, then the other share is also $|e|$-close to the red area, and therefore one party rounding up and the other party rounding down always yields the correct result (as long as $|e| < \frac{q}{4}$). More precisely, assume that t_0 is in the red area and $\mathsf{Round}'(0, t_0) = 0$, i.e., $t_0 \in \left[\frac{q}{4} \pm |e|\right]$ (the other cases are similar). Now, if $t_0 + t_1 = e \mod q$, then $t_1 \in \left[-\frac{q}{4} \pm 2 \cdot |e|\right]$, and thus $\mathsf{Round}'(1, t_1) = 0$. If $t_0 + t_1 = \frac{q}{2} + e \mod q$, on the other hand, it holds $t_1 \in \left[\frac{q}{4} \pm |e|\right]$ and thus $\mathsf{Round}'(1, t_1) = 1$ as required.

Given these two observations, we obtain our first core lemma (Lemma 1). We present the corresponding rounding procedures in Figure 3. Here, $\mathcal{P}_0$ always follows a fixed rounding procedure, where $\mathcal{P}_0$ uses the normal rounding procedure *outside* the red area, and the rounding procedure $\mathsf{RoundDown}$ *inside* the red area. If its share is within the red area, it sets $\mathsf{flag} = 1$ for the corresponding wire, and $\mathsf{flag} = 0$ otherwise. If the share of $\mathcal{P}_1$ is *outside* the (now larger) red area, it follows the standard rounding procedure, and sets $\mathsf{flag} = 0$. If the share of $\mathcal{P}_1$ is *inside* the larger red area, it follows both the standard rounding procedure (depicted by the blue arrows) and the $\mathsf{RoundUp}$ rounding procedure (depicted by the

dashed arrows) and sets the flags to 0 and 1, respectively. For reconstruction, the parties resort to the alternative ("dashed") computation path whenever both parties set $\mathsf{flag} = 1$ on the corresponding wire.
Together with our new lifting lemma, this yields our HSS scheme. A crucial part of our construction is carefully taking account of the gates with $\mathsf{flag} = 1$, which we explain in the main body.

2 Preliminaries

In this section we define the HSS primitive as well as the computational model for programs supported by our construction. We begin by introducing some notation. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$. We denote by λ the security parameter.
We will work with the ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, where $N \leq \mathrm{poly}(\lambda)$ is a power of 2. The infinity norm on $\mathcal{R}$ is defined as $\|x\|_\infty = \max_{i \in [n]} |x_i|$ for $x \in \mathcal{R}$ with coefficients $x_1, \ldots, x_n$. For $q \in \mathbb{N}$, let $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$, where we consider elements of $\mathcal{R}_q$ to have all their coefficients in the interval $(-q/2, \ldots, q/2]$.

2.1 Homomorphic Secret Sharing

We consider homomorphic secret sharing with a general decoding algorithm for the reconstruction of shares, as defined by Boyle et al. [13], in the public-key setting. We note that HSS is commonly defined with the stronger requirement of additive reconstruction, which enjoys several useful properties. By considering the more general definition, our scheme is able to forego some of those properties for efficiency. Moreover, we show that the decoding functionality can be easily and securely realized, depending on the application setting.

Definition 1 (Homomorphic Secret Sharing). *A 2-party public-key homomorphic secret sharing (HSS) scheme for a class of programs $\mathcal{P}$ consists of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ with the following syntax:*

- $\mathsf{Gen}(1^\lambda)$: *On input a security parameter 1^λ, the key generation algorithm outputs a public key pk and a pair of evaluation keys $(\mathsf{ek}_0, \mathsf{ek}_1)$.*
- $\mathsf{Enc}(\mathsf{pk}, x)$: *On input the public key pk and an input value x, the encryption algorithm outputs a ciphertext $\mathbf{c}$.*
- $\mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, (\mathbf{c}_1, \ldots, \mathbf{c}_n), P, \beta)$: *On input a party index $\sigma \in \{0, 1\}$, evaluation key ek_σ, a vector of n ciphertexts, a program $P \in \mathcal{P}$ with n input values, and an output modulus β, the homomorphic evaluation algorithm outputs a share y_σ.*
- $\mathsf{Dec}(y_0, y_1, \beta)$: *On input shares y_0, y_1 and an output modulus β, the decoding algorithm outputs a value y.*

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

Perfect correctness. *For all $\lambda \in \mathbb{N}$, inputs $x_1, \ldots, x_n$, program $P \in \mathcal{P}$, and integer $\beta \geq 2$, we have*

$$\mathsf{Dec}(y_0, y_1, \beta) = P(x_1, \ldots, x_n),$$

where $(\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{Gen}(1^\lambda)$, $\mathbf{c}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, x_i)$ *for* $i \in [n]$ *and* $y_\sigma \leftarrow \mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, (\mathbf{c}_1, \ldots, \mathbf{c}_n), P, \beta)$ *for* $\sigma \in \{0,1\}$.

Security. *For all* $\lambda \in \mathbb{N}$ *and for all PPT adversaries* $\mathcal{A}$,

$$\Pr\left[\mathcal{A}(\mathsf{state}, \mathsf{pk}, \mathsf{ek}_\sigma, \mathbf{c}) = b \;\middle|\; \begin{array}{l}(\sigma, x_0, x_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda) \\ b \leftarrow \{0,1\} \\ (\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathbf{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, x_b)\end{array}\right] - \frac{1}{2} \leq \mathsf{negl}(\lambda).$$

Remark 1. We relax the definition of HSS by not requiring the Eval algorithm to run in polynomial time, but only *expected* polynomial time, which will be the case in our construction. This can be converted into polynomial time by halting the computation after some fixed number of steps.

2.2 Restricted Multiplication Straight-line Programs

Our HSS scheme supports homomorphic evaluation of the class of Restricted Multiplication Straight-line (RMS) programs. These are a restricted form of arithmetic circuits in which multiplication of intermediate values is not possible; only multiplication of an input value by an intermediate value (or *memory value*) is allowed.

Definition 2 (RMS programs).
An RMS program over the ring $\mathcal{R}$ *consists of a magnitude bound* B_{max} *and a sequence of instructions of the four types below, each indicating its ingoing and outgoing wires and ordered by a unique identifier* $\mathsf{id} \in \mathbb{N}$.

- *Load input into memory: instruction* $(\mathsf{load}, \mathsf{id}, x, w)$ *sets input* x *as a memory value in wire* w *(*$\hat{y}_w \leftarrow \hat{x}$*).*
- *Add values in memory: instruction* $(\mathsf{add}, \mathsf{id}, u, v, w)$ *adds the values in wires* u *and* v *(*$\hat{y}_w \leftarrow \hat{y}_u + \hat{y}_v$*).* [5]
- *Multiply input by memory value: instruction* $(\mathsf{mult}, \mathsf{id}, x, v, w)$ *multiplies the input* x *and the memory value in wire* v *(*$\hat{y}_w \leftarrow \hat{x} \cdot \hat{y}_v$*).*
- *Output from memory: instruction* $(\mathsf{out}, \mathsf{id}, w)$ *outputs the value in wire* w *as an element of* $\mathcal{R}_\beta$.

If at any step of execution the magnitude of a memory value exceeds the bound B_{max} *(i.e.* $\|\hat{y}_w\|_\infty > B_{\mathsf{max}}$*), the output of the program on the corresponding input is defined to be* $\perp$*. Otherwise the output is the sequence of values given by the* out *instruction.*
We define the multiplicative size *of an RMS program* P *as its number of* load *and* mult *instructions, and we denote it by* $|P|$.

[5] We assume that for every instruction $(\mathsf{add}, \mathsf{id}, u, v, w)$ such that u (resp. v) is the output wire of a previous instruction with id id_u (resp. id_v) we have $\mathsf{id}_u < \mathsf{id}_v$. This ensures that shares corresponding to u are computed before shares corresponding to v in our evaluation algorithm.

Note the distinction between the magnitude bound B_{max} and the output modulus β. For example, in an RMS program computing a Boolean function $f : \{0,1\}^k \to \{0,1\}$, the input values 0 and 1 would be interpreted as integers, B_{max} would be a bound on the greatest integer appearing as the result of an operation, and the output modulus would be $\beta = 2$. Our HSS scheme will require $\beta \leq B_{\mathsf{max}} < p < q$, where p and q are, respectively, the plaintext modulus and ciphertext modulus of the underlying encryption scheme.

Remark 2. The definition of RMS program in [14] includes an additional operation type which allows input values to be added. The class of functions computable with this additional operation is the same, but it allows some functions to be computed using fewer multiplications, which may result in a more efficient homomorphic evaluation. We omit this operation from our definition, but we note that our HSS also supports it, in identical fashion to the BKS scheme. In both constructions this feature requires adjusting the bound on the ciphertext noise according to the maximum number of input additions, which influences the parameters of the scheme.

3 The Homomorphic Secret Sharing Scheme

In this section, we describe our homomorphic secret sharing scheme. Our HSS is an adaptation of the BKS scheme [14]. It supports homomorphic evaluations of the same class of functions: *Restricted Multiplication Straight-Line* (RMS) programs. Informally, we adapt the original BKS scheme by incorporating a new error reconciliation procedure. The protocol parameters of the BKS scheme are chosen such that correctness errors only occur with negligible probability. By contrast, our reconciliation procedure allows for smaller protocol parameters, since potential errors occurring during the homomorphic evaluations are corrected by the error reconciliation procedure. As a result the internal protocol parameters can to chosen to be polynomial in the security parameter, whereas BKS scheme requires superpolynomial protocol parameters, thereby reducing the communication complexity.

3.1 The Protocol

Both the BKS scheme and our adaptation crucially rely on a public-key encryption scheme $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ with *nearly linear decryption*, i.e., for all key-pairs $(\mathsf{pk}, \mathbf{s}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$, messages $m \in \mathbb{Z}_p$ and ciphertexts $\mathbf{c} \leftarrow \mathsf{PKE.Enc}_{\mathsf{pk}}(m)$, it holds that

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

for some "small" noise term $|e| \leq B_{\mathsf{err}}$.
Since the PKE has nearly linear decryption, the decryption procedure simply rounds the inner-product $\langle \mathbf{c}, \mathbf{s} \rangle$ of the ciphertext and the secret

Fig. 4: Homomorphic Secret Sharing - Key Generation

$\mathsf{HSS.Gen}(1^\lambda)$: Generate $(\mathsf{pk}, \mathbf{s}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$ and sample a PRF key $K \leftarrow \{0,1\}^\lambda$ uniformly at random. Sample $\mathbf{s}_0 \leftarrow \mathbb{Z}_q^d$ and define

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

Output $(\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1)$, where $\mathsf{ek}_0 := (K, \mathbf{s}_0), \mathsf{ek}_1 := (K, \mathbf{s}_1) \in \{0,1\}^\lambda \times \mathbb{Z}_q^d$.

key, multiplied by $0 < p/q < 1$, to the nearest integer, i.e.,

$$\mathsf{PKE.Dec}(\mathbf{c}, \mathbf{s}) = \left\lceil \frac{p}{q} \cdot \langle \mathbf{c}, \mathbf{s} \rangle \right\rfloor \mod p\,.$$

We assume that the first coefficient of the secret key $\mathbf{s} \in \mathbb{Z}^d$ equals 1. This property is crucially required by the HSS construction, and it is satisfied by most PKE schemes with nearly linear decryption.
Further, for simplicity, we assume PKE to be defined over $\mathbb{Z}$. For this reason, our homomorphic secret sharing scheme will also be defined over $\mathbb{Z}$. However, all techniques and results have a straightforward generalization to rings of the form $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ for N a power of 2, namely, the rounding and lifting procedures are applied to each of the N coordinates of elements of $\mathcal{R}$.
As shown in [14], if PKE has nearly linear decryption and pseudorandom ciphertexts, there exists a *Key Dependent Message* (KDM) oracle PKE.OKDM that, without knowledge of the secret key, outputs encryptions of scalar multiples of the secret key ([14], Lemma 3). More precisely, for all $j \in \{1, \ldots, d\}$ and $x \in \mathbb{Z}$,

$$\mathbf{c}_j \leftarrow \mathsf{PKE.OKDM}(\mathsf{pk}, x, j) \quad \text{s.t.} \quad \langle \mathbf{c}_j, \mathbf{s} \rangle = x \cdot s_j + e \mod q\,,$$

where $\mathbf{s} = (s_1, \ldots, s_d)$ and $|e| \leq B_{\mathsf{err}}$. By linearity, the KDM oracle allows encryptions of arbitrary linear combinations of the secret key to be generated.
Let us now continue to describe our 2-party homomorphic secret sharing scheme HSS. Besides a PKE scheme with the above properties, the HSS construction also requires a keyed pseudorandom function PRF. The key-generation of our HSS scheme, described in Figure 4, is identical to that of the BKS scheme. The HSS public key is simply a public key for the PKE scheme and each evaluation key contains an additive secret share $\mathbf{s}_\sigma$ of the secret key together with a PRF key K.
The second functionality of the HSS is encryption. It allows parties to encrypt the inputs to the RMS program that is to be evaluated. However, an HSS encryption of an input value $x \in \mathbb{Z}$ is different from a standard PKE encryption of x. Instead, it is an encryption of the key-dependent vector $x \cdot \mathbf{s} \in \mathbb{Z}_q^d$, where $\mathbf{s} \in \mathbb{Z}_q^d$ is the secret key corresponding to the public key pk generated in the key generation. Hence, the HSS encryption

Fig. 5: Homomorphic Secret Sharing - Encryption

$\mathsf{HSS.Enc}(\mathsf{pk}, x)$: Compute $\mathbf{c}_j \leftarrow \mathsf{PKE.OKDM}(\mathsf{pk}, x, j)$ for $j = 1, \ldots, d$. Output the ciphertext $\mathbf{C} := (\mathbf{c}_1, \ldots, \mathbf{c}_d)$.

of x is a vector of d PKE encryptions, each to a different key-dependent message $x \cdot s_i$ for $i \in \{1, \ldots, d\}$. Note that, since $\mathbf{s} = (1, s_2, \ldots, s_d) \in \mathbb{Z}_q^d$, the first component of an HSS encryption is a standard PKE encryption of $x \cdot 1 = x$. The HSS encryption functionality, again identical to the one used by the BKS scheme, is described in Figure 5. Intuitively, security of our HSS scheme follows from the security of OKDM and from each share $\mathbf{s}_\sigma$ individually hiding $\mathbf{s}$.

The reason for using this "key-dependent" encryption is that, by deploying a distributed decryption, the two parties can take encrypted input values and obtain additive secret shares of the vector $x \cdot \mathbf{s}$. The BKS scheme shows how to perform certain operations on secret shares of key-dependent messages of this form. More precisely, it shows that the following operations can be performed *locally* (i.e., without requiring interaction between the two parties):

- **Addition**: given a secret share of $x \cdot \mathbf{s}$ and a secret share $y \cdot \mathbf{s}$, obtain a secret share of $(x + y) \cdot \mathbf{s}$.
- **Multiplication by Input Value**: given an HSS encryption of x and a secret share of $y \cdot \mathbf{s}$, obtain a secret share of $xy \cdot \mathbf{s}$.

The HSS scheme thus distinguishes between (encrypted) input values and intermediate computation values, also referred to as *memory* values. The above functionalities immediately imply an HSS for RMS programs.

Our scheme deviates from BKS in how it performs the above HSS operations. In the BKS scheme these operations involve a distributed decryption, which in turn involves the rounding of a noisy value followed by a "lifting" of shares mod p to shares mod q. Both of these steps may fail, causing a correctness error, and the BKS scheme chooses its parameters such that such errors only occur with negligible probability. In our approach, we employ procedures Round and Lift (defined in Section 3.2) which indicate whether an error may have occurred and correct it if necessary.

In more detail, for party $\mathcal{P}_0$, the output of Round is of the form $(\mathsf{flag}_0, z_0) \in \{0, 1\} \times \mathbb{Z}_p$. If $\mathsf{flag}_0 = 0$ (no error can occur), then z_0 is obtained by rounding as usual, while if $\mathsf{flag}_0 = 1$ (an error may occur), then z_0 is the result of an alternative "error-correcting" rounding.

Before describing the procedure for party $\mathcal{P}_1$, note that, since the parties cannot communicate, there is no guarantee that their flags will coincide. Moreover, the error-correcting requires the two parties to be in sync, i.e. correctness is not guaranteed if one party follows the usual rounding and the other the alternative rounding. Therefore, it may seem necessary that each party computes both the usual and alternative values when their flag is positive, in order to use one of them depending on the flag of the

Fig. 6: Homomorphic Secret Sharing - Evaluation for party $\mathcal{P}_0$

$\mathsf{HSS.Eval}(0, \mathsf{ek}_0, (\mathbf{C}^1, \ldots, \mathbf{C}^n), P, \beta)$: Parse $\mathsf{ek}_0 = (K, \mathbf{s}_0)$ and P as a sequence of RMS instructions. Initialize pos_0 as the empty binary string. Proceed as follows for each instruction in P.

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

$$(\mathbf{t}_0^w, \mathsf{pos}_0) \leftarrow \mathsf{Mult}_0(K, \mathsf{id}, \mathbf{C}, \mathbf{s}_0, \mathsf{pos}_0)\,,$$

  where Mult_0 is the algorithm described in Figure 8.
- *Add values in memory:* On instruction $(\mathsf{add}, \mathsf{id}, u, v, w)$, set

$$\mathbf{t}_0^w := \mathbf{t}_0^u + \mathbf{t}_0^v \mod q\,.$$

- *Multiply input by memory value:* On instruction $(\mathsf{mult}, \mathsf{id}, \mathbf{C}, v, w)$, compute

$$(\mathbf{t}_0^w, \mathsf{pos}_0) \leftarrow \mathsf{Mult}_0(K, \mathsf{id}, \mathbf{C}, \mathbf{t}_0^v, \mathsf{pos}_0)\,,$$

  where Mult_0 is the algorithm described in Figure 8.
- *Output from memory:* On instruction $(\mathsf{out}, \mathsf{id}, w)$, parse $\mathbf{t}_0^w$ as $\mathbf{t}_0^w = (x_0, \hat{\mathbf{t}}_0)$ for some $x_0 \in \mathbb{Z}_q, \hat{\mathbf{t}}_0 \in \mathbb{Z}_q^{d-1}$ and output

$$y_0 := (H(\mathsf{pos}_0), x_0 \bmod \beta),$$

  where $H(a_1, \ldots, a_k) = \{i \in [k] \,|\, a_i = 1\}$.

other party. However, we are able to define Round in a way such that whenever $\mathsf{flag}_0 = 1$ we have $\mathsf{flag}_1 = 1$ as well. This allows us to define Round for $\mathcal{P}_0$ as described above, always computing a single value z_0, and have only $\mathcal{P}_1$ compute two different values when $\mathsf{flag}_1 = 1$.
For $\mathcal{P}_1$, Round either outputs $\mathsf{flag}_1 = 0$ and z_1, or $\mathsf{flag}_1 = 1$ and (z_1, z_1'), where z_1 and z_1' denote the outputs of the usual and alternative rounding, respectively. The following table displays the 3 different scenarios that may occur, and whether the parties should use corrected values or not.

		Flag of $\mathcal{P}_0$	
		0	1
Flag of $\mathcal{P}_1$	0	No Correction	———
	1	No Correction	Error Correction

Similarly, errors can occur and be mitigated in the so-called lifting step, which always follows rounding.
The homomorphic evaluation procedure for party $\mathcal{P}_0$ is presented in Figure 6. For every wire w in the RMS program P we compute a vector $\mathbf{t}_0^w \in \mathbb{Z}_q^d$ which is $\mathcal{P}_0$'s additive share of $x^w\mathbf{s}$, where x^w is the value

Fig. 7: Homomorphic Secret Sharing - Evaluation for party $\mathcal{P}_1$

$\mathsf{HSS.Eval}(1, \mathsf{ek}_1, (\mathbf{C}^1, \ldots, \mathbf{C}^n), P, \beta)$: Parse $\mathsf{ek}_1 = (K, \mathbf{s}_1)$ and P as a sequence of RMS instructions. Initialize L_1 as an empty list. Proceed as follows for each instruction in P.

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

$$(T_1^w, L_1) \leftarrow \mathsf{Mult}_1(K, \mathsf{id}, \mathbf{C}, \{(\epsilon, \mathbf{s}_1)\}, L_1),$$

  where Mult_1 is the algorithm described in Figure 9 and ϵ denotes the empty binary string.
- *Add values in memory:* On instruction $(\mathsf{add}, \mathsf{id}, u, v, w)$, set

$$T_1^w := \Big\{(\mathsf{pos}_1,\ \mathbf{t}_1^u + \mathbf{t}_1^v \mod q) \,\Big|\, \mathsf{pos}_1 \in L_1, (\mathsf{pos}_1^u, \mathbf{t}_1^u) \in T_1^u, \\ (\mathsf{pos}_1^v, \mathbf{t}_1^v) \in T_1^v, \mathsf{pos}_1^u \subseteq \mathsf{pos}_1^v \subseteq \mathsf{pos}_1\Big\}.$$

- *Multiply input by memory value:* On instruction $(\mathsf{mult}, \mathsf{id}, \mathbf{C}, v, w)$, compute

$$(T_1^w, L_1) \leftarrow \mathsf{Mult}_1(K, \mathsf{id}, \mathbf{C}, T_1^v, L_1),$$

  where Mult_1 is the algorithm described in Figure 9.
- *Output from memory:* On instruction $(\mathsf{out}, \mathsf{id}, w)$, output the list

$$y_1 := \Big\{(H(\mathsf{pos}),\ x_1 \bmod \beta) \,\Big|\, (\mathsf{pos}, \mathbf{t}_1) \in T_1^w, \\ \mathbf{t}_1 = (x_1, \hat{\mathbf{t}}_1), x_1 \in \mathbb{Z}_q, \hat{\mathbf{t}}_1 \in \mathbb{Z}_q^{d-1}\Big\},$$

  where $H(a_1, \ldots, a_k) = \{i \in [k] \,|\, a_i = 1\}$.

of P at w. Throughout this algorithm we keep track of the variable $\mathsf{pos}_0 \in \{0,1\}^*$ which denotes the sequence of flags of $\mathcal{P}_0$. After each "multiplicative" operation (i.e. load or mult instruction), the flags generated during that operation are appended to the string $\mathsf{pos}_0 \in \{0,1\}^*$. Adding a pseudorandom value $\mathsf{PRF}(K, \mathsf{id})$ before each rounding step guarantees that the shares are always close to uniform, and therefore the occurrences of positive flags are independent from one instruction to another. Finally, the output of Eval consists of a compression $H(\mathsf{pos}_0)$ of the flag sequence of $\mathcal{P}_0$ and the first component of $\mathbf{t}_0^w$, which is an additive share of $P(x_1, \ldots, x_n)$. The compression function H simply outputs the list of indices with a flag set to 1 (which will be constant in number). The use of H is crucial in obtaining succinct output shares, as the size of pos_0 is proportional to the size $|P|$ of the program.

In Figure 7 we present the homomorphic evaluation procedure for party $\mathcal{P}_1$, which is similar to that of $\mathcal{P}_0$ but has an added degree of complexity, since $\mathcal{P}_1$ generates two different possible values for its additive share whenever it gets a positive flag, and must keep track of all possible combinations. The global variable L_1 in this algorithm is the list of binary

Fig. 8: Algorithm Mult_0, employed by party $\mathcal{P}_0$ on loading and multiplication instructions.

Input $(K, \mathsf{id}, \mathbf{C}, \mathbf{t}, \mathsf{pos})$

Parse $\mathbf{C} = (\mathbf{c}_1, \ldots, \mathbf{c}_d)$
For each $i \in [d]$:
    $(\mathsf{flag}_i, z_i) \leftarrow \mathsf{Round}(0, \langle \mathbf{t}, \mathbf{c}_i \rangle + \mathsf{PRF}(K, (\mathsf{id}, i)) \mod q)$
    $(\mathsf{flag}'_i, v_i) \leftarrow \mathsf{Lift}(0, z_i)$
$\mathbf{t}' \leftarrow (v_1, \ldots, v_d)$
$\mathsf{pos}' \leftarrow \mathsf{pos}||\mathsf{flag}_1||\mathsf{flag}'_1|| \ldots ||\mathsf{flag}_d||\mathsf{flag}'_d$

Output $(\mathbf{t}', \mathsf{pos}')$

Fig. 9: Algorithm Mult_1, employed by party $\mathcal{P}_1$ on loading and multiplication instructions.

Input $(K, \mathsf{id}, \mathbf{C}, T, L)$

Parse $\mathbf{C} = (\mathbf{c}_1, \ldots, \mathbf{c}_d)$, $T = ((\mathsf{pos}_1, \mathbf{t}_1), \ldots, (\mathsf{pos}_\ell, \mathbf{t}_\ell))$
For each $(i, j) \in [d] \times [\ell]$:
    $V_{ij} \leftarrow \emptyset$
    $(\mathsf{flag}_{ij}, z^0_{ij}, z^1_{ij}) \leftarrow \mathsf{Round}(1, \langle \mathbf{t}_j, \mathbf{c}_i \rangle - \mathsf{PRF}(K, (\mathsf{id}, i)) \mod q)$
    $(\mathsf{flag}^0_{ij}, v^{00}_{ij}, v^{01}_{ij}) \leftarrow \mathsf{Lift}(1, z^0_{ij})$
    $V_{ij} \leftarrow V_{ij} \cup \{(00, v^{00}_{ij})\}$
    If $\mathsf{flag}^0_{ij} = 1$:
        $V_{ij} \leftarrow V_{ij} \cup \{(01, v^{01}_{ij})\}$
    If $\mathsf{flag}_{ij} = 1$:
        $(\mathsf{flag}^1_{ij}, v^{10}_{ij}, v^{11}_{ij}) \leftarrow \mathsf{Lift}(1, z^1_{ij})$
        $V_{ij} \leftarrow V_{ij} \cup \{(10, v^{10}_{ij})\}$
        If $\mathsf{flag}^1_{ij} = 1$:
            $V_{ij} \leftarrow V_{ij} \cup \{(11, v^{11}_{ij})\}$
$T' \leftarrow \Big\{ \big(\mathsf{pos}||a_1|| \ldots ||a_d,\ (v_1, \ldots, v_d)\big) \Big| j \in [\ell],\ \mathsf{pos} \in L,$
$\mathsf{pos}_j \subseteq \mathsf{pos},\ (a_i, v_i) \in V_{ij} \Big\}$
$L' \leftarrow \Big\{ \mathsf{pos} \Big| (\mathsf{pos}, \mathbf{t}) \in T' \Big\}$

Output (T', L')

Fig. 10: Homomorphic Secret Sharing - Decoding

$\mathsf{HSS.Dec}(y_0, y_1, \beta)$: Parse the shares as $y_0 = (u_0, x_0)$ and $y_1 = \{(u_1^{(1)}, x_1^{(1)}), \ldots, (u_1^{(k)}, x_1^{(k)})\}$. Output $x_0 + x_1^{(i)} \mod \beta$, where i is the unique index such that $u_0 = u_1^{(i)}$.

strings which includes all possible sequences of flags of $\mathcal{P}_0$ – recall that whenever $\mathcal{P}_1$ has $\mathsf{flag}_1 = 0$ it knows that $\mathsf{flag}_0 = 0$, but if $\mathsf{flag}_1 = 1$ then flag_0 can be either 0 or 1. To each wire w in P we associate a list T_1^w of pairs of the form $(\mathsf{pos}_1, \mathbf{t}_1^w)$, where $\mathbf{t}_1^w$ is the additive share corresponding to the value of P at w and pos_1 is the corresponding sequence of flags. The output of the evaluation algorithm for $\mathcal{P}_1$ is a list of pairs of the same form as the output for $\mathcal{P}_0$, one for each possible flag sequence.
Finally, in the decoding algorithm, depicted in Figure 10, we identify the additive shares x_0, x_1 which correspond to the same sequence of flags and add them to obtain $P(x_1, \ldots, x_n)$.

Remark 3. We omit an optimization step consisting of checking if the two values associated with a positive flag for $\mathcal{P}_1$ are the same, which provides a reduction of the flag probability by a factor of 2 in both rounding and lifting.

Remark 4. Like the BKS scheme, our protocol can also be converted into a secret-key HSS version, which is more efficient for those applications which do not require the public-key capabilities.

3.2 Rounding and Lifting

Below we present our rounding procedure and analyse its properties. The corresponding step in the BKS protocol consists of multiplying the share $v \in \mathbb{Z}_q$ by p/q and rounding it to the nearest integer to obtain a share in $\mathbb{Z}_p$. This introduces a correctness error with probability proportional to p/q (see Lemma 7). Our approach solves this issue by flagging instances in which an error could occur if both parties were to round their shares to the nearest integer and correcting it by having one party round up and the other round down in those instances.
Recall that we consider the representation $\mathbb{Z}_n = \{-\lceil (n-1)/2 \rceil, \ldots, \lfloor (n-1)/2 \rfloor\}$ for any $n \in \mathbb{N}$. We first define the operations $\mathsf{RoundDown}$, $\mathsf{RoundUp}$ and $\mathsf{RoundNear}$, which map a value v from $\mathbb{Z}_q$ to $\mathbb{Z}_p$ by scaling and then rounding it down, up, or to the nearest integer, respectively:

$$\begin{aligned} \mathsf{RoundDown}(v) &= \lfloor (p/q) \cdot v \rfloor \mod p, \\ \mathsf{RoundUp}(v) &= \lceil (p/q) \cdot v \rceil \mod p, \\ \mathsf{RoundNear}(v) &= \lceil (p/q) \cdot v \rfloor \mod p. \end{aligned}$$

The deterministic procedure Round, which takes as input a party identifier $\sigma \in \{0,1\}$ and a value $v \in \mathbb{Z}_q$, is defined as follows:

$$\mathsf{Round}(0,v) = \begin{cases} (1, \mathsf{RoundDown}(v)), & \text{if } v \in \mathsf{bad}_{B_{\mathsf{err}}}, \\ (0, \mathsf{RoundNear}(v)), & \text{otherwise,} \end{cases}$$

$$\mathsf{Round}(1,v) = \begin{cases} (1, \mathsf{RoundNear}(v), \mathsf{RoundUp}(v)), & \text{if } v \in \mathsf{bad}_{2B_{\mathsf{err}}}, \\ (0, \mathsf{RoundNear}(v), \bot), & \text{otherwise,} \end{cases}$$

where $\mathsf{bad}_{B_{\mathsf{err}}} = \left\{ v \in \mathbb{Z}_q \,\middle|\, |v \mod (q/p)| \geq q/(2p) - B_{\mathsf{err}} \right\}$ and $\mathsf{bad}_{2B_{\mathsf{err}}}$ is analogously defined.

Lemma 5 (Rounding correctness). *Let $p, q, B_{\mathsf{err}} \in \mathbb{N}$ be such that q is a multiple of p and $B_{\mathsf{err}} < q/(4p)$. Then, for any $v_0, v_1 \in \mathbb{Z}_q$, $m \in \mathbb{Z}_p$ and $e \in \mathbb{Z}$ such that $|e| \leq B_{\mathsf{err}}$ and*

$$v_0 + v_1 = (q/p) \cdot m + e \mod q,$$

the outputs $(\mathsf{flag}_0, z_0) \leftarrow \mathsf{Round}(0, v_0)$, $(\mathsf{flag}_1, z_1, z_1') \leftarrow \mathsf{Round}(1, v_1)$ satisfy the following:
(i) If $\mathsf{flag}_0 = 0$, then $z_0 + z_1 = m \mod p$.
(ii) If $\mathsf{flag}_0 = 1$, then $\mathsf{flag}_1 = 1$ and $z_0 + z_1' = m \mod p$.

Proof. Let v_0, v_1, m, e be such that $v_0 + v_1 = (q/p) \cdot m + e \mod q$ and $|e| \leq B_{\mathsf{err}}$, and let $(\mathsf{flag}_0, z_0) \leftarrow \mathsf{Round}(0, v_0)$, $(\mathsf{flag}_1, z_1, z_1') \leftarrow \mathsf{Round}(1, v_1)$. To prove the first claim, assume that $\mathsf{flag}_0 = 0$. Then there exist $k, r \in \mathbb{Z}$ such that $v_0 = (q/p) \cdot k + r$ and $|r| < q/(2p) - B_{\mathsf{err}}$. Therefore

$$v_1 = (q/p) \cdot (m - k) + e - r \mod q$$

and $|e - r| \leq |e| + |r| < q/(2p)$. It follows that

$$z_0 = \lceil (p/q) \cdot v_0 \rfloor = \lceil k + \underbrace{(p/q) \cdot r}_{\in (-1/2, 1/2)} \rfloor = k \mod p,$$

$$z_1 = \lceil (p/q) \cdot v_1 \rfloor = \lceil m - k + \underbrace{(p/q) \cdot (e - r)}_{\in (-1/2, 1/2)} \rfloor = m - k \mod p,$$

which shows that $z_0 + z_1 = m \mod p$.
We now prove the second claim. If $\mathsf{flag}_0 = 1$, there exist $k, r \in \mathbb{Z}$ such that $v_0 = (q/p) \cdot k + r$ and $q/(2p) - B_{\mathsf{err}} \leq r \leq q/(2p) + B_{\mathsf{err}}$. The other share is then $v_1 = (q/p) \cdot (m-k) + e - r \mod q$, where $q/(2p) - 2B_{\mathsf{err}} \leq e - r \leq q/(2p) + 2B_{\mathsf{err}}$, since $|e| \leq B_{\mathsf{err}}$. Therefore $|v_1 \mod (q/p)| \geq q/(2p) - 2B_{\mathsf{err}}$ and $\mathsf{flag}_1 = 1$. Moreover, observe that $e < r$ and $(p/q) \cdot (r - e) < 1$, since

$$r \leq q/(2p) + B_{\mathsf{err}} < q/p - B_{\mathsf{err}} \leq q/p + e.$$

It follows that

$$z_0 = \lfloor (p/q) \cdot v_0 \rfloor = \lfloor k + \underbrace{(p/q) \cdot r}_{\in [0,1)} \rfloor = k \mod p,$$

$$z_1' = \lceil (p/q) \cdot v_1 \rceil = \lceil m - k + \underbrace{(p/q) \cdot (e - r)}_{\in (-1, 0]} \rceil = m - k \mod p,$$

and therefore $z_0 + z_1' = m \mod p$. □

Lemma 6 (Rounding flag probability). *Let $p, q, B_{\mathsf{err}} \in \mathbb{N}$ be such that q is a multiple of p and $B_{\mathsf{err}} < q/(4p)$. Let $v_0, v_1 \in \mathbb{Z}_q$ be uniformly random subject to*

$$v_0 + v_1 = (q/p) \cdot m + e \mod q,$$

where $m \in \mathbb{Z}_p$ and $|e| \leq B_{\mathsf{err}}$ are fixed. Let also $(\mathsf{flag}_1, z_1, z_1') \leftarrow \mathsf{Round}(1, v_1)$. Then

$$\Pr[\mathsf{flag}_1 = 1 \text{ and } z_1 \neq z_1'] = 2B_{\mathsf{err}} \cdot (p/q).$$

Proof. Let $u_1 = v_1 \mod (q/p)$ and note that u_1 is uniformly distributed in $\mathbb{Z}_{q/p}$. Recall that $\mathsf{flag}_1 = 1$ if and only if $|u_1| \geq q/(2p) - 2B_{\mathsf{err}}$. Moreover, $\mathsf{RoundNear}(v_1) \neq \mathsf{RoundUp}(v_1)$ if and only if the fractional part of $(p/q) \cdot v_1$ is in the interval $(0, 1/2)$, which holds if and only if $0 < u_1 < q/(2p)$. Define the set

$$S = \left\{ u \in \mathbb{Z}_{q/p} \,\middle|\, q/(2p) - 2B_{\mathsf{err}} \leq u < q/(2p) \right\}.$$

If $q/p = 2k + 1$ for some $k \in \mathbb{N}$, then $S = \{k - 2B_{\mathsf{err}} + 1, \ldots, k\}$, while if $q/p = 2k$ then $S = \{k - 2B_{\mathsf{err}}, \ldots, k - 1\}$. In both cases $|S| = 2B_{\mathsf{err}}$. Therefore $\Pr[\mathsf{flag}_1 = 1 \text{ and } z_1 \neq z_1'] = \Pr[u_1 \in S] = |S| \cdot (p/q) = 2B_{\mathsf{err}} \cdot (p/q)$. □

Lemma 7 (Rounding error probability). *Let $p, q, B_{\mathsf{err}} \in \mathbb{N}$ be such that q is a multiple of p and $B_{\mathsf{err}} < q/(4p)$. Let $v_0, v_1 \in \mathbb{Z}_q$ be random subject to*

$$v_0 + v_1 = (q/p) \cdot m + e \mod q,$$

where $m \in \mathbb{Z}_p$ and $|e| \leq B_{\mathsf{err}}$ are fixed. Then

$$\Pr[\mathsf{RoundNear}(v_0) + \mathsf{RoundNear}(v_1) \neq m \mod p] \geq (|e| - 1) \cdot (p/q).$$

Proof. Define $u_\sigma = v_\sigma \mod (q/p)$, for $\sigma = 0, 1$, and assume first that $e < 0$. Observe that, if $u_0, u_1 \in (0, q/(2p))$, then a rounding error occurs: since $e = u_0 + u_1 \mod (q/p)$ and $-(q/p) < e < 0$, it must be the case that $e = u_0 + u_1 - (q/p)$, and therefore $\mathsf{RoundNear}(v_0) + \mathsf{RoundNear}(v_1) = m - 1$. If $q/p = 2k + 1$ for some $k \in \mathbb{N}$, then

$$\Pr[u_0, u_1 \in (0, q/(2p))] = \Pr[u_0 \in \{k + e + 1, \ldots, k\}] = |e| \cdot (p/q).$$

Alternatively, if $q/p = 2k$, then

$$\Pr[u_0, u_1 \in (0, q/(2p))] = \Pr[u_0 \in \{k + e + 1, \ldots, k - 1\}] = (|e| - 1) \cdot (p/q).$$

By a similar reasoning it can be seen that in the case $e \geq 0$ a rounding error occurs with probability at least $|e| \cdot (p/q)$, if q/p is odd, or $(|e| + 1) \cdot (p/q)$, if q/p is even.

□

Now we present the lifting procedure, which always follows rounding. In the BKS protocol this step is simply an inclusion: a share $z \in \mathbb{Z}_q$ becomes $z \in \mathbb{Z}_p$. However, as shown in Lemma 10, a correctness error occurs with probability proportional to $1/p$. Again, our new procedure overcomes

this issue by predicting and correcting possible errors to guarantee that additive shares modulo p are always converted into shares modulo q of the same secret value.
The deterministic procedure Lift, which takes as input a party identifier $\sigma \in \{0,1\}$ and a value $z \in \mathbb{Z}_p$, is defined as follows:

$$\mathsf{Lift}(0,z) = \begin{cases} (1,z), & \text{if } z \in \mathsf{bad}^+_{B_{\mathsf{max}}}, \\ (1,z+p), & \text{if } z \in \mathsf{bad}^-_{B_{\mathsf{max}}}, \\ (0,z), & \text{otherwise,} \end{cases}$$

$$\mathsf{Lift}(1,z) = \begin{cases} (1,z,z-p), & \text{if } z \in \mathsf{bad}^+_{2B_{\mathsf{max}}}, \\ (1,z,z), & \text{if } z \in \mathsf{bad}^-_{2B_{\mathsf{max}}}, \\ (0,z,\perp), & \text{otherwise,} \end{cases}$$

where $\mathsf{bad}^+_{B_{\mathsf{max}}} = [p/2 - B, p/2)$, $\mathsf{bad}^-_{B_{\mathsf{max}}} = [-p/2, -p/2 + B]$, and $\mathsf{bad}^+_{2B_{\mathsf{max}}}$, $\mathsf{bad}^-_{2B_{\mathsf{max}}}$ are analogously defined. The proofs of the following three lemmas can be found in the full version of this paper.

Lemma 8 (Lifting correctness). *Let $p, B_{\mathsf{max}} \in \mathbb{N}$ be such that $B_{\mathsf{max}} < p/6$. Then, for any $z_0, z_1 \in \mathbb{Z}_p$, $m \in \mathbb{Z}$ such that $|m| \leq B_{\mathsf{max}}$ and*

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

the outputs $(\mathsf{flag}_0, v_0) \leftarrow \mathsf{Lift}(0, z_0)$, $(\mathsf{flag}_1, v_1, v_1') \leftarrow \mathsf{Lift}(1, z_1)$ satisfy the following:
(i) If $\mathsf{flag}_0 = 0$, then $v_0 + v_1 = m$ over $\mathbb{Z}$.
(ii) If $\mathsf{flag}_0 = 1$, then $\mathsf{flag}_1 = 1$ and $v_0 + v_1' = m$ over $\mathbb{Z}$.

Lemma 9 (Lifting flag probability). *Let $p, B_{\mathsf{max}} \in \mathbb{N}$ be such that $B_{\mathsf{max}} < p/6$. Let $z_0, z_1 \in \mathbb{Z}_p$ be random subject to*

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

where $m \in \mathbb{Z}_p$. Let also $(\mathsf{flag}_1, v_1, v_1') \leftarrow \mathsf{Lift}(1, z_1)$. Then

$$\Pr[\mathsf{flag}_1 = 1 \text{ and } v_1 \neq v_1'] = 2B_{\mathsf{max}}/p.$$

Lemma 10 (Lifting error probability). *Let $p, B_{\mathsf{max}} \in \mathbb{N}$ be such that $B_{\mathsf{max}} < p/6$. Let $z_0, z_1 \in \mathbb{Z}_p$ be random subject to*

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

where $m \in \mathbb{Z}_p$. Then

$$\Pr[z_0 + z_1 \neq m] \geq (|m| - 1)/p.$$

We can now prove our main result.

Theorem 1 (HSS correctness and security). *Let PKE be a public-key encryption scheme with plaintext space $\mathcal{R}_p$ and ciphertext space $\mathcal{R}_q^d$, satisfying the properties of nearly linear decryption (with error bound B_{err}) and pseudorandom ciphertexts, such that $B_{\mathsf{err}} < q/(4p)$. Let also PRF be a pseudorandom function taking values in $\mathcal{R}_q$. Then the 2-party homomorphic secret sharing scheme described in Figures 4 to 10 is perfectly correct and secure, as per Definition 1, and supports homomorphic evaluation of polynomial-sized RMS programs with magnitude bound B_{max} and output modulus β such that $\beta \leq B_{\mathsf{max}} < p/6$.*

Proof. Security follows immediately from the security of the BKS HSS [14], as the algorithms Gen and Enc are identical in the two schemes and the security definition is independent of the Eval algorithm. Note that this is a consequence of KDM security and of the fact that the evaluation keys individually hide the secret encryption key.

We will now show that our scheme satisfies perfect correctness. Let $y_0 = (H(\mathsf{pos}_0), z_0)$ and $y_1 = \{(H(\mathsf{pos}_1^{(1)}), z_1^{(1)}), \ldots, (H(\mathsf{pos}_1^{(k)}), z_1^{(k)})\}$ be the evaluated shares corresponding to an RMS program P on input $x_1, \ldots, x_n$.

Observe that, according to the definition of the algorithms Mult_0 and Mult_1, there always exists $i^* \in [k]$ such that $\mathsf{pos}_1^{(i^*)} = \mathsf{pos}_0$. This follows from the fact that, at any rounding or lifting step with position tag pos, party $\mathcal{P}_1$ always computes a value associated to $\mathsf{pos}||0$ and, by part (ii) of Lemmas 5 and 8, if $\mathcal{P}_0$ has a value associated to $\mathsf{pos}||1$ then so does $\mathcal{P}_1$. Furthermore, the index i^* is unique, since the binary strings $\mathsf{pos}_1^{(j)}$ are all distinct. Since the compression function H is injective, the only index i such that $H(\mathsf{pos}_1^{(i)}) = H(\mathsf{pos}_0)$ is i^*.

We will show below that, during homomorphic evaluation of P, for all wires w we have

$$\mathbf{t}_0^w + \mathbf{t}_1^w = x^w \mathbf{s} \mod q \tag{1}$$

whenever $(\mathsf{pos}_1, \mathbf{t}_1^w) \in T_1^w$ and $\mathsf{pos}_1^w = \mathsf{pos}_0^w$, where pos_0^w is the flag sequence pos_0 of $\mathcal{P}_0$ at the time wire w is evaluated, $x^w \in \mathcal{R}$ denotes the value of P at w and $\mathbf{s} = (1, \hat{\mathbf{s}}) \in \mathcal{R} \times \mathcal{R}^{d-1}$ is the PKE secret key.

The final output will be $\mathsf{Dec}(y_0, y_1, \beta) = z_0 + z_1^{(i^*)} \mod \beta$, where $(z_0, \hat{\mathbf{t}}_0) = \mathbf{t}_0^w$, $(z_1^{(i^*)}, \hat{\mathbf{t}}_1) = \mathbf{t}_1^w$, $(\mathsf{pos}_1^{(i^*)}, \mathbf{t}_1^w) \in T_1^w$ for an output wire w and $\mathsf{pos}_1^{(i^*)} = \mathsf{pos}_0$. If equation (1) holds, then by looking only at the first component of each vector in the equation we see

$$z_0 + z_1^{(i^*)} = x^w \cdot 1 = P(x_1, \ldots, x_n) \mod q,$$

hence $\mathsf{Dec}(y_0, y_1, \beta) = P(x_1, \ldots, x_n)$ with probability 1. [6]

It remains only to check that equation (1) holds for every instruction in P of type load, add or mult.

- For instruction $(\mathsf{load}, \mathsf{id}, (\mathbf{c}_1, \ldots, \mathbf{c}_d), w)$, where $\mathbf{c}_i \leftarrow \mathsf{PKE.OKDM}(\mathsf{pk}, y, i)$, by the nearly linear decryption property we have

$$\begin{aligned}(\mathbf{t}_0^w)_i + (\mathbf{t}_1^w)_i &= \langle \mathbf{s}_0, \mathbf{c}_i \rangle + \mathsf{PRF}(K, (\mathsf{id}, i)) + \langle \mathbf{s}_1, \mathbf{c}_j \rangle - \mathsf{PRF}(K, (\mathsf{id}, i)) \\ &= \langle \mathbf{s}, \mathbf{c}_i \rangle = (q/p) \cdot y \cdot s_i + e_i \mod q\end{aligned}$$

  for some $|e_i| \leq B_{\mathsf{err}}$. [7] We can thus apply Lemma 5 followed by Lemma 8 to conclude that, for the matching flags (i.e. $\mathsf{pos}_1^w = \mathsf{pos}_0^w$), the corresponding shares $\mathbf{t}_0^w, \mathbf{t}_1^w$ satisfy $\mathbf{t}_0^w + \mathbf{t}_1^w = y\,\mathbf{s} \mod q$.

[6] We assume here that β divides q, so that shares mod q are also shares mod β. If we wish to avoid this assumption, we can simply perform a lifting step to obtain shares over $\mathbb{Z}$ before reducing them mod β.

[7] Here we again consider the case $\mathcal{R} = \mathbb{Z}$ for simplicity. For $\mathcal{R}$ of dimension N, the equation applies to each coordinate of y.

– For instruction $(\mathsf{add}, \mathsf{id}, u, v, w)$, assume equation (1) holds for $(\mathbf{t}_0^u, \mathbf{t}_1^u)$ and $(\mathbf{t}_0^v, \mathbf{t}_1^v)$, where $\mathsf{pos}_1^u \subseteq \mathsf{pos}_1^v \subseteq \mathsf{pos}_1^w$ and $\mathsf{pos}_0^\tau = \mathsf{pos}_1^\tau$ for $\tau \in \{u, v, w\}$. Then

$$\mathbf{t}_0^w + \mathbf{t}_1^w = \mathbf{t}_0^u + \mathbf{t}_0^v + \mathbf{t}_1^u + \mathbf{t}_1^v = x^u\mathbf{s} + x^v\mathbf{s} = x^w\mathbf{s} \mod q.$$

– For instruction $(\mathsf{mult}, \mathsf{id}, (\mathbf{c}_1, \ldots, \mathbf{c}_d), v, w)$, assuming equation (1) holds for $(\mathbf{t}_0^v, \mathbf{t}_1^v)$ we have

$$\begin{aligned}(\mathbf{t}_0^w)_i + (\mathbf{t}_1^w)_i &= \langle \mathbf{t}_0^v, \mathbf{c}_i\rangle + \mathsf{PRF}(K, (\mathsf{id}, i)) + \langle \mathbf{t}_1^v, \mathbf{c}_j\rangle - \mathsf{PRF}(K, (\mathsf{id}, i)) \\ &= x^v\langle \mathbf{s}, \mathbf{c}_i\rangle = (q/p)x^v \cdot y \cdot s_i + e_i \mod q\end{aligned}$$

and as in the load instruction we conclude that $\mathbf{t}_0^w + \mathbf{t}_1^w = x^v\, y\, \mathbf{s} \mod q$.

□

3.3 Impossibility of Local Share Conversion

The next theorem shows that the local share conversion procedure that lies at the heart of lattice-based HSS cannot achieve perfect correctness with additive reconstruction. Therefore one must either allow correctness error (which can only be made negligible with a superpolynomial modulus) or relax the requirement for reconstruction.

Theorem 2 (Correctness error of share conversion). *Let $m \in \mathbb{Z}_p$ and $e \in D$, where $\{0, 1, -1\} \subseteq D \subseteq (-q/(2p), q/(2p))$. Let also $v_0, v_1 \in \mathbb{Z}_q$ be sampled uniformly subject to*

$$v_0 + v_1 = (q/p) \cdot m + e \mod q.$$

Then, for any local share conversion functions $g_0, g_1 : \mathbb{Z}_q \to \mathbb{Z}_q$, there exist $m \in \mathbb{Z}_p$ and $e \in D$ such that

$$\Pr[g_0(v_0) + g_1(v_1) \neq m \mod q] \geq p/(3q).$$

Proof. We show that in each interval $I_k \subseteq \mathbb{Z}_q$ of the form $I_k := [k \cdot q/p, (k+1) \cdot q/p)$ there exists $v_0 \in I_k$ such that an error $g_0(v_0) + g_1(v_1) \neq m$ occurs for at least one of the pairs $(m, e) := (0, 0)$, $(m, e) := (1, -1)$ or $(m, e) := (0, 1)$. Since there are p disjoint intervals I_k, one of these three choices of (m, e) must have at least $p/3$ values v_0 in the above conditions and the result follows from the fact that v_0 is uniform.
To prove the above claim, consider $v_0 := k \cdot q/p$ and $v_1 := -v_0$. If $g_0(v_0)+g_1(v_1) \neq 0$ we have found an error for $(m, e) := (0, 0)$, as $v_0+v_1 = 0$ and $v_0 \in I_k$. Meanwhile, $v_0' := (k+1)\cdot q/p - 1$ satisfies $v_0'+v_1 = q/p \cdot 1 - 1$, hence if $g_0(v_0') + g_1(v_1) \neq 1$ we have found an error for $(m, e) := (1, -1)$. Suppose now that $g_0(v_0) + g_1(v_1) = 0$ and $g_0(v_0') + g_1(v_1) = 1$. Then $g_0(v_0) \neq g_0(v_0')$ and there must exist $\tilde{v}_0 \in [v_0, v_0')$ such that $g_0(\tilde{v}_0) \neq g_0(\tilde{v}_0 + 1)$. Note that $\tilde{v}_0, \tilde{v}_0 + 1 \in I_k$. Then, unless an error occurs with $\tilde{v}_0$ and $(m, e) := (0, 0)$ or $\tilde{v}_0 + 1$ and $(m, e) := (0, 1)$, by taking $\tilde{v}_1 := -\tilde{v}_0$ we obtain

$$g_0(\tilde{v}_0) + g_1(\tilde{v}_1) = g_0(\tilde{v}_0 + 1) + g_1(\tilde{v}_1) = 0,$$

since $\tilde{v}_0 + \tilde{v}_1 = 0$ and $(\tilde{v}_0 + 1) + \tilde{v}_1 = 1$. This contradicts the assumption $g_0(\tilde{v}_0) \neq g_0(\tilde{v}_0 + 1)$. □

4 Efficiency and Parameters

In this section we compute concrete parameters for our HSS scheme and compare them with the BKS scheme [14]. The next lemma gives us an expression for the average number of elements of the list that constitutes the share y_1 of party $\mathcal{P}_1$ after evaluating a program P. We are then able to choose parameters such that this number is bounded by a constant. Since the running time of the evaluation algorithm of $\mathcal{P}_1$ is proportional to this quantity, the lemma also implies that it runs in expected polynomial time.

Lemma 11 (Expected share size). *Consider the HSS scheme described above, with ciphertext space $\mathcal{R}_q^d$, where $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$. Let P be an RMS program of multiplicative size $|P|$. Denote by p_{round}, p_{lift} the probabilities of party $\mathcal{P}_1$ having a positive flag in a single rounding or lifting step, respectively. Then the expected total number E of terminal values in the homomorphic evaluation of P by $\mathcal{P}_1$ is*

$$E = \big((1 + p_{\mathsf{round}})(1 + p_{\mathsf{lift}})\big)^{dN|P|}.$$

We defer the proof of Lemma 11 to the full version. As a consequence of Lemmas 6, 9 and 11, we obtain the following bound, which we can use to choose parameters for the HSS scheme:

$$E \leq \big((1 + 2B_{\mathsf{err}}p/q)(1 + 2B_{\mathsf{max}}/p)\big)^{dN|P|}.$$

We instantiate PKE with the Ring-LWE based encryption scheme of Lyubashevsky, Peikert and Regev [27] over the ring $\mathcal{R} = \mathbb{Z}[X]/(X^N+1)$, giving us $B_{\mathsf{err}} = 1$, $d = 2$. Then, if we wish to bound the expected number of terminal values E by some value $\gamma > 1$, setting $p \geq 8B_{\mathsf{max}}N|P|/\ln\gamma$ and $q \geq 8pN|P|/\ln\gamma$ gives

$$E \leq (1 + \ln\gamma/(4N|P|))^{4N|P|} \leq \gamma,$$

which justifies that γ is indeed an upper bound. For instance, if we choose $\gamma = 2$, party $\mathcal{P}_1$ will have, on average, a single positive flag throughout the homomorphic evaluation and two terminal values on which to perform reconstruction.

In Tables 3 and 4 we present parameters of our scheme in this RLWE instantiation, namely the ring dimension N and the ciphertext modulus q, when we choose the bound $\gamma = 2$ for the expected number of terminal values and maximum program sizes 2^{10} and 2^{20}, respectively. These are given in function of the magnitude bound B_{max} of plaintexts during the computation. For comparison, Table 5 shows the parameters for the corresponding instantiation of the BKS HSS scheme. We observe that our scheme reduces the size of the modulus q by nearly a factor of 2 for programs with up to 2^{20} operations (or by a greater factor, if we further restrict the program size) while also reducing N by a factor of 2 and attaining similarly high estimated computational security.

The security estimates on Tables 3 – 5 were obtained by computing, for magnitude bound B_{max}, the smallest pair (N, q) with at least 80 bits

B_{max}	N	$\log q$	Security
2	2048	51	147.3
2^{16}	2048	66	109.4
2^{32}	2048	82	86.0
2^{64}	4096	116	122.9
2^{128}	8192	182	159.5
2^{256}	8192	310	89.1

Table 3: HSS parameters for $|P| = 2^{10}$, $\gamma = 2$.

B_{max}	N	$\log q$	Security
2	2048	71	100.9
2^{16}	2048	86	81.6
2^{32}	4096	104	139.0
2^{64}	4096	136	103.0
2^{128}	8192	202	141.7
2^{256}	8192	330	83.6

Table 4: HSS parameters for $|P| = 2^{20}$, $\gamma = 2$.

B_{max}	N	$\log q$	Security
2	4096	137	103.3
2^{16}	4096	167	83.7
2^{32}	8192	203	142.0
2^{64}	8192	267	104.9
2^{128}	16384	399	143.9
2^{256}	16384	655	84.6

Table 5: BKS HSS parameters, with error probability 2^{-40}.

of computational security, as predicted by the lattice estimator tool of Albrecht et al. [1]. Note that the parameters of the BKS HSS scheme are also dependent on the size of the program P. The parameters on Table 5 correspond to a correctness error probability of 2^{-40} for *each* (multiplicative) operation in P.

The parameter γ can be adjusted to reduce the frequency of raised flags for a relatively small cost in the size of lattice parameters. For instance, setting $\gamma = 1.01$ boosts the probability that there are no raised flags in the entire computation to at least $1 - (\gamma - 1) = 0.99$, at the price of increasing the modulus q by a factor of $(\ln 2 / \ln 1.01)^2 \approx 2^{12}$, compared to the choice $\gamma = 2$.

On the other hand, since the size of the input shares is much larger than the size of the output shares, it can make sense to choose larger parameter γ for certain applications. One should note though, that if the parties wish to execute linear postprocessing on the output shares (e.g., a counting query over a large database), then the total expected number of shares scales with 2^{γ}.

5 Applications

Our scheme retains most of the standard applications of HSS, even without having the usual property of additive reconstruction. It is particularly suited for scenarios where there is asymmetry between the parties performing the computation (e.g. two servers of different sizes).

Private database queries. We explore in detail one of the applications of the BKS HSS scheme and show that our construction provides an overall improvement in efficiency. A 2-server private database query protocol involves two non-colluding servers, each holding a copy of a public database DB, and a client, who can issue queries on the database. The protocol should allow the client to obtain the answer of its query while hiding both the query and the answer from the servers. HSS gives a simple solution to this problem with only one round of communication: in this protocol the client sends an encryption of its query to both servers, who then homomorphically compute shares of the answer and return them to the client. HSS for branching programs supports many expressive queries, such as conjunctive keyword search and pattern matching.

Remark 5. Unlike with secure 2-party computation, in this setting there are no concerns with the security of the reconstruction procedure. We can simply have both servers send their shares to the client, who evaluates the decoding algorithm directly with minimal computational cost.

Linear post-processing of shares. There are scenarios in which the additive reconstruction property of other HSS schemes is quite useful, such as when computing a counting query. This type of query returns the number of elements of the database satisfying some predicate Q, which can be written as $\sum_{x \in \mathsf{DB}} Q(x)$, where $Q(x) = 1$ if x satisfies Q and $Q(x) = 0$ otherwise. Because of this additive representation, instead of homomorphically evaluating the query on the database at once, the servers can evaluate the predicate individually on each database element. The shares q_σ^x corresponding to each element x can then be locally summed to obtain $q_\sigma = \sum_{x \in \mathsf{DB}} q_\sigma^x$ and this value sent to the client, who recovers the result of the query as $q_0 + q_1 \mod \beta$. In the BKS HSS scheme, this approach allows using the optimal case of $B_{\mathsf{max}} = 2$ on the individual HSS evaluations, even though the query output is not bounded by 2.
Although our construction does not benefit from the additive reconstruction property, we can employ a similar technique and show that even in this setting we obtain a performance improvement. Recall that, in our scheme, a share evaluated by party $\mathcal{P}_0$ is of the form (u, q) where u is the compression of the flag sequence of $\mathcal{P}_0$ and q is the additive share of the result, while a share evaluated by $\mathcal{P}_1$ is a list of pairs of the same form. $\mathcal{P}_0$ can homomorphically evaluate Q on each $x \in \mathsf{DB}$ to obtain (u_0^x, q_0^x) and then send $y_0 = (u_0^{x_1}, \ldots, u_0^{x_M}, q_0 := \sum_{x \in \mathsf{DB}} q_0^x)$ as its final share to the client, where $M = |\mathsf{DB}|$ is the database size. Similarly, $\mathcal{P}_1$ obtains M lists from evaluating Q on every database element and its output to the client is a list of shares of the same form as y_0, one for each possible choice of a single element from each of the M lists. The client can then reconstruct by summing q_0 and the corresponding value from $\mathcal{P}_1$'s share. This solution may look terribly inefficient for the fact that the size of $\mathcal{P}_1$'s output is proportional to the product of the number of elements of all M lists, but we can set the probability of any list having more than one element to be very low.

A concrete example. Consider a database DB with entries of the form (x, W_x) where x is a document and W_x is a list of keywords. Given a target list of keywords W, we wish to count the number of documents containing all the keywords in W. That is, we consider a counting query for the predicate $Q_W(x, W_x) = 1$ if $W \subseteq W_x$. Suppose the database size is $M = 1024$, the client's query consists of 4 keywords, and each document has 10 keywords with 128 bits of length. This can be achieved by an RMS program P with around $|P| = 5120$ multiplications. For this application the BKS scheme requires as parameters $N = 4096$ and $\log q = 137$, which gives a share size of $3N \log q \approx 210$kB for each input bit, for a total of 107MB of communication to each server. In our scheme, choosing $\gamma = 1.0001$ allows us to use $N = 2048$, $\log q = 81$. This results in an input share size of 60.7kB and a total of 31MB sent from client to server. Since the (expected) size of the compressed flag sequence is $|H(\mathsf{pos})| =$

$\gamma \log |\mathsf{pos}|$ and the output modulus of the query should be $\beta = M$, the size of the first output share is $|y_0| = M\gamma \log(4N|P|) + \log \beta \approx 3.2$kB and the size of the second output share is $|y_1| = \gamma^M |y_0| \approx 3.5$kB. Meanwhile, the output share size in BKS is only $\log \beta \approx 1.2B$ for both servers. Note that only a single output share is sent from each server to the client, so the bulk of communication lies in the input sharing step for both approaches.

A drawback of our solution is that the size of the output shares grows with the size of the database (linearly for $\mathcal{P}_0$, and exponentially with base γ for $\mathcal{P}_1$). However, the communication bottleneck is still the size of the input shares and not of the output, as illustrated in the example above. For more general queries, for which this technique relying on additive reconstruction is not applicable, our scheme again provides an improvement over BKS HSS in both computation and communication costs.

Acknowledgments. Thomas Attema was supported by the Vraaggestuurd Programma Cyber Security & Resilience, part of the Dutch Top Sector High Tech Systems and Materials program. Pedro Capitão and Lisa Kohl have been supported by the NWO Gravitation project QSC.

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Cryptology ePrint Archive, Report 2015/046 (2015), `https://eprint.iacr.org/2015/046`
2. Alwen, J., Krenn, S., Pietrzak, K., Wichs, D.: Learning with rounding, revisited - new reduction, properties and applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 57–74. Springer, Heidelberg (Aug 2013)
3. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (Aug 2009)
4. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 719–737. Springer, Heidelberg (Apr 2012)
5. Bogdanov, A., Guo, S., Masny, D., Richelson, S., Rosen, A.: On the hardness of learning with rounding over small modulus. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016-A, Part I. LNCS, vol. 9562, pp. 209–224. Springer, Heidelberg (Jan 2016)
6. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 896–912. ACM Press (Oct 2018)
7. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019,

Part III. LNCS, vol. 11694, pp. 489–518. Springer, Heidelberg (Aug 2019)
8. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Orrù, M.: Homomorphic secret sharing: Optimizations and applications. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 2105–2122. ACM Press (Oct / Nov 2017)
9. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg (Apr 2015)
10. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg (Aug 2016)
11. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 1292–1303. ACM Press (Oct 2016)
12. Boyle, E., Gilboa, N., Ishai, Y.: Group-based secure computation: Optimizing rounds, communication, and computation. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 163–193. Springer, Heidelberg (Apr / May 2017)
13. Boyle, E., Gilboa, N., Ishai, Y., Lin, H., Tessaro, S.: Foundations of homomorphic secret sharing. In: Karlin, A.R. (ed.) ITCS 2018. vol. 94, pp. 21:1–21:21. LIPIcs (Jan 2018)
14. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 3–33. Springer, Heidelberg (May 2019)
15. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled PSI from fully homomorphic encryption with malicious security. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 1223–1237. ACM Press (Oct 2018)
16. Chillotti, I., Orsini, E., Scholl, P., Smart, N.P., Van Leeuwen, B.: Scooby: Improved multi-party homomorphic secret sharing based on fhe. In: International Conference on Security and Cryptography for Networks. pp. 540–563. Springer (2022)
17. Chor, B., Gilboa, N., Naor, M.: Private information retrieval by keywords. Citeseer (1997)
18. Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled PSI from homomorphic encryption with reduced computation and communication. pp. 1135–1150. ACM Press (2021)
19. Corrigan-Gibbs, H., Boneh, D., Mazières, D.: Riposte: An anonymous messaging system handling millions of users. In: 2015 IEEE Symposium on Security and Privacy. pp. 321–338. IEEE Computer Society Press (May 2015)
20. Couteau, G., Meyer, P.: Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 842–870. Springer, Heidelberg (Oct 2021)

21. Dinur, I., Keller, N., Klein, O.: An optimal distributed discrete log protocol with applications to homomorphic secret sharing. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 213–242. Springer, Heidelberg (Aug 2018)
22. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 93–122. Springer, Heidelberg (Aug 2016)
23. Fazio, N., Gennaro, R., Jafarikhah, T., Skeith III, W.E.: Homomorphic secret sharing from paillier encryption. In: Okamoto, T., Yu, Y., Au, M.H., Li, Y. (eds.) ProvSec 2017. LNCS, vol. 10592, pp. 381–399. Springer, Heidelberg (Oct 2017)
24. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (May 2014)
25. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Designs, Codes and Cryptography 75(3), 565–599 (2015)
26. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (May / Jun 2010)
27. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 35–54. Springer, Heidelberg (May 2013)
28. Orlandi, C., Scholl, P., Yakoubov, S.: The rise of paillier: Homomorphic secret sharing and public-key silent OT. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 678–708. Springer, Heidelberg (Oct 2021)
29. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005)
30. Roy, L., Singh, J.: Large message homomorphic secret sharing from DCR and applications. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 687–717. Springer, Heidelberg, Virtual Event (Aug 2021)
31. Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery 22(11), 612–613 (Nov 1979)
32. Wang, F., Yun, C., Goldwasser, S., Vaikuntanathan, V., Zaharia, M.: Splinter: Practical private queries on public data. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). pp. 299–313 (2017)