Multiparty Computation from Somewhat Homomorphic Encryption

Ivan Damgård[1], Valerio Pastro[1], Nigel Smart[2], and Sarah Zakarias[1]

[1] Department of Computer Science, Aarhus University
[2] Department of Computer Science, Bristol University

Abstract. We propose a general multiparty computation protocol secure against an active adversary corrupting up to $n-1$ of the n players. The protocol may be used to compute securely arithmetic circuits over any finite field $\mathbb{F}_{p^k}$. Our protocol consists of a preprocessing phase that is both independent of the function to be computed and of the inputs, and a much more efficient online phase where the actual computation takes place. The online phase is unconditionally secure and has total computational (and communication) complexity linear in n, the number of players, where earlier work was quadratic in n. Moreover, the work done by each player is only a small constant factor larger than what one would need to compute the circuit in the clear. We show this is optimal for computation in large fields. In practice, for 3 players, a secure 64-bit multiplication can be done in 0.05 ms. Our preprocessing is based on a somewhat homomorphic cryptosystem. We extend a scheme by Brakerski et al., so that we can perform distributed decryption and handle many values in parallel in one ciphertext. The computational complexity of our preprocessing phase is dominated by the public-key operations, we need $O(n^2/s)$ operations per secure multiplication where s is a parameter that increases with the security parameter of the cryptosystem. Earlier work in this model needed $\Omega(n^2)$ operations. In practice, the preprocessing prepares a secure 64-bit multiplication for 3 players in about 13 ms.

1 Introduction

A central problem in theoretical cryptography is that of secure multiparty computation (MPC). In this problem n parties, holding private inputs $x_1, \ldots, x_n$, wish to compute a given function $f(x_1, \ldots, x_n)$. A protocol for doing this securely should be such that honest players get the correct result and this result is the only new information released, even if some subset of the players is controlled by an adversary.

In the case of *dishonest majority*, where more than half the players are corrupt, unconditionally secure protocols cannot exist. Under computational assumptions, it was shown in [6] how to construct UC-secure MPC protocols that handle the case where all but one of the parties are actively corrupted. The public-key machinery one needs for this is typically expensive so efficient solutions are hard to design for dishonest majority. Recently, however, a new approach has been proposed making such protocols more practical. This approach

works as follows: one first designs a general MPC protocol in the *preprocessing model*, where access to a "trusted dealer" is assumed. The dealer does not need to know the function to be computed, nor the inputs, he just supplies raw material for the computation before it starts. This allows the "online" protocol to use only cheap information theoretic primitives and hence be efficient. Finally, one implements the trusted dealer by a secure protocol using public-key techniques, this protocol can then be run in a preprocessing phase. The current state of the art in this respect are the protocols in Bendlin et al., Damgård/Orlandi and Nielsen et al. [3, 9, 16]. The "MPC-in-the-head" technique of Ishai et al. [13, 12] has similar overall asymptotic complexity, but larger constants and a less efficient online phase.

Recently, another approach has become possible with the advent of Fully Homomorphic Encryption (FHE) by Gentry [10]. In this approach all parties first encrypt their input under the FHE scheme; then they evaluate the desired function on the ciphertexts using the homomorphic properties, and finally they perform a distributed decryption on the final ciphertexts to get the results. The advantage of the FHE-based approach is that interaction is only needed to supply inputs and get output. However, the low bandwidth consumption comes at a price; current FHE schemes are very slow and can only evaluate small circuits, i.e., they actually only provide what is known as somewhat homomorphic encryption (SHE). This can be circumvented in two ways; either by assuming circular security and implementing an expensive bootstrapping operation, or by extending the parameter sizes to enable a "levelled FHE" scheme which can evaluate circuits of large degree (exponential in the number of levels) [4]. The main cost, much like other approaches, is in terms of the number of multiplications in the arithmetic circuit. So whilst theoretically appealing the approach via FHE is not competitive in practice with the traditional MPC approach.

1.1 Contributions of this paper.

Optimal Online Phase. We propose an MPC protocol in the preprocessing model that computes securely an arithmetic circuit C over any finite field $\mathbb{F}_{p^k}$. The protocol is statistically UC-secure against active and adaptive corruption of up to $n-1$ of the n players, and we assume synchronous communication and secure point-to-point channels. Measured in elementary operations in $\mathbb{F}_{p^k}$ the total amount of work done is $O(n \cdot |C| + n^3)$ where $|C|$ is the size of C. All earlier work in this model had complexity $\Omega(n^2 \cdot |C|)$. A similar improvement applies to the communication complexity and the amount of data one needs to store from the preprocessing. Hence, the work done by each player in the online phase is essentially independent of n. Moreover, it is only a small constant factor larger than what one would need to compute the circuit in the clear. This is the first protocol in the preprocessing model with these properties[3].

[3] With dishonest majority, successful termination cannot be guaranteed, so our protocols simply abort if cheating is detected. We do not, however, identify *who* cheated, indeed the standard definition of secure function evaluation does not require this.

Finally, we show a lower bound implying that w.r.t the amount of data required from the preprocessing, our protocol is optimal up to a constant factor. We also obtain a similar lower bound on the number of bit operations required, and hence the computational work done in our protocol is optimal up to poly-logarithmic factors.

All results mentioned here hold for the case of large fields, i.e., where the desired error probability is $(1/p^k)^c$, for a small constant c. Note that many applications of MPC need integer arithmetic, modular reductions, conversion to binary, etc., which we can emulate by computing in $\mathbb{F}_p$ with p large enough to avoid overflow. This naturally leads to computing with large fields. As mentioned, our protocol works for all fields, but like earlier work in this model it is less efficient for small fields by a factor of essentially $\lceil \frac{\mathsf{sec}}{\log p^k} \rceil$ for error probability $2^{-\Theta(\mathsf{sec})}$, see the full version for details.

Obtaining our result requires new ideas compared to [3], which was previously state of the art and was based on additive secret sharing where each share in a secret is authenticated using an information theoretic Message Authentication Code (MAC). Since each player needs to have his own key, each of the n shares need to be authenticated with n MACs, so this approach is inherently quadratic in n. Our idea is to authenticate the secret value itself instead of the shares, using a single global key. This seems to lead to a "chicken and egg" problem since one cannot check a MAC without knowing the key, but if the key is known, MACs can be forged. Our solution to this involves secret sharing the key as well, carefully timing when values are revealed, and various tricks to reduce the amortized cost of checking a set of MACs.

Efficient use of FHE for MPC. As a conceptual contribution we propose what we believe is "the right" way to use FHE/SHE for *computationally* efficient MPC, namely to use it for implementing a preprocessing phase. The observation is that since such preprocessing is typically based on the classic circuit randomization technique of Beaver [2], it can be done by evaluating in parallel many small circuits of small multiplicative depth (in fact depth 1 in our case). Thus SHE suffices, we do not need bootstrapping, and we can use the SHE SIMD approach of [17] to handle many values in parallel in a single ciphertext.

To capitalize on this idea, we apply the SIMD approach to the cryptosystem from [5] (see also [11] where this technique is also used). To get the best performance, we need to do a non-trivial analysis of the parameter values we can use, and we prove some results on norms of embeddings of a cyclotomic field for this purpose. We also design a distributed decryption procedure for our cryptosystem. This protocol is only robust against passive attacks. Nevertheless, this is sufficient for the overall protocol to be actively secure. Intuitively, this is because the only damage the adversary can do is to add a known error term to the decryption result obtained. The effect of this for the online protocol is that certain shares of secret values may be incorrect, but this will caught by the

Identification of cheaters is possible but we do not know how to do this while maintaining complexity linear in n.

check involving the MACs. Finally we adapt a zero-knowledge proof of plaintext knowledge from [3] for our purpose and in particular we improve the analysis of the soundness guarantees it offers. This influences the choice of parameters for the cryptosystem and therefore improves overall performance.

An Efficient Preprocessing Protocol. As a result of the above, we obtain a constant-round preprocessing protocol that is UC-secure against active and static corruption of $n-1$ players assuming the underlying cryptosystem is semantically secure, which follows from the polynomial (PLWE) assumption. UC-security for dishonest majority cannot be obtained without a set-up assumption. In this paper we assume that a key pair for our cryptosystem has been generated and the secret key has been shared among the players.

Whereas previous work in the preprocessing/online model [3, 9] use $\Omega(n^2)$ public-key operations per secure multiplication, we only need $O(n^2/s)$ operations, where s is a number that grows with the security parameter of the SHE scheme (we have $s \approx 12000$ in our concrete instantiation for computing in $\mathbb{F}_p$ where $p \approx 2^{64}$). We stress that our adapted scheme is exactly as efficient as the basic version of [5] that does not allow this optimization, so the improvement is indeed "genuine".

In comparison to the approach mentioned above where one uses FHE throughout the protocol, our combined preprocessing and online phase achieves a result that is incomparable from a theoretical point of view, but much more practical: we need more communication and rounds, but the computational overhead is much smaller – we need $O(n^2/s \cdot |C|)$ public key operations compared to $O(n \cdot |C|)$ for the FHE approach, where for realistic values of n and s, we have $n^2/s \ll n$. Furthermore, we only need a low depth SHE which is much more efficient in the first place. And finally, we can push all the work using SHE into a, function independent, preprocessing phase.

Performance in practice. Both the preprocessing and online phase have been implemented and tested for 3 players on up-to-date machines connected on a LAN. The preprocessing takes about 13 ms amortized time to prepare one multiplication in $\mathbb{F}_p$ for a 64-bit p, with security level corresponding roughly to 1024 bit RSA and an error probability of 2^{-40} for the zero-knowledge proofs (the error probability can be lowered to 2^{-80} by repeating the ZK proofs which will at most double the time). This is 2-3 orders of magnitude faster than preliminary estimates for the most efficient instantiation of [3]. The online phase executes a secure 64-bit multiplication in 0.05 ms amortized time. These rough orders of magnitude, and the ability to deal with a non-trivial number of players, are born out by a recent implementation of the protocols described in this paper [7].

Concurrent Related Work. In recent independent work [15, 1, 11], Meyers at al., Asharov et al. and Gentry et al. also use an FHE scheme for multiparty computation. They follow the pure FHE approach mentioned above, using a threshold decryption protocol tailored to the specific FHE scheme. They focus primarily on round complexity, while we want to minimize the computational overhead.

We note that in [11], Gentry et al. obtain small overhead by showing a way to use the FHE SIMD approach for computing any circuit homomorphically. However, this requires full FHE with bootstrapping (to work on arbitrary circuits) and does not (currently) lead to a practical protocol.

In [16], Nielsen et al. consider secure computing for Boolean Circuits. Their online phase is similar to that of [3], while the preprocessing is a clever and very efficient construction based on Oblivious Transfer. This result is complementary to ours in the sense that we target computations over large fields which is good for some applications whereas for other cases, Boolean Circuits are the most compact way to express the desired computation. Of course, one could use the preprocessing from [16] to set up data for our online phase, but current benchmarks indicate that our approach is faster for large fields, say of size 64 bits or more.

We end the introduction by covering some basic notation which will be used throughout this paper. For a vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ we denote by $\|\mathbf{x}\|_\infty := \max_{1 \leq i \leq n} |x_i|$, $\|\mathbf{x}\|_1 := \sum_{1 \leq i \leq n} |x_i|$ and $\|\mathbf{x}\|_2 := \sqrt{\sum |x_i|^2}$. We let $\epsilon(\kappa)$ denote an unspecified negligible function of κ. If S is a set we let $x \leftarrow S$ denote assignment to the variable x with respect to a uniform distribution on S; we use $x \leftarrow s$ for a value s as shorthand for $x \leftarrow \{s\}$. If A is an algorithm $x \leftarrow A$ means assign to x the output of A, where the probability distribution is over the random coins of A. Finally $x := y$ means "x is defined to be y".

2 Online Protocol

Our aim is to construct a protocol for arithmetic multiparty computation over $\mathbb{F}_{p^k}$ for some prime p. More precisely, we wish to implement the ideal functionality $\mathcal{F}_{\text{AMPC}}$, presented in . Our MPC protocol is structured in a preprocessing (or offline) phase and an online phase. We start out in this section by presenting the online phase which assumes access to an ideal functionality $\mathcal{F}_{\text{PREP}}$. In Section 5 we show how to implement this functionality in an independent preprocessing phase.

In our specification of the online protocol, we assume for simplicity that a broadcast channel is available at unit cost, that each party has only one input, and only one public output value is to be computed. In the full version we explain how to implement the broadcasts we need from point-to-point channels and lift the restriction on the number of inputs and outputs without this affecting the overall complexity.

Before presenting the concrete online protocol we give the intuition and motivation behind the construction. We will use unconditionally secure MACs to protect secret values from being manipulated by an active adversary. However, rather than authenticating shares of secret values as in [3], we authenticate the shared value itself. More concretely, we will use a global key α chosen randomly in $\mathbb{F}_{p^k}$, and for each secret value a, we will share a additively among the players, and we also secret-share a MAC αa. This way to represent secret values is linear, just like the representation in [3], and we can therefore do secure multi-

plication based on multiplication triples à la Beaver [2] that we produce in the preprocessing.

An immediate problem is that opening a value reliably seems to require that we check the MAC, and this requires players know α. However, as soon as α is known, MACs on other values can be forged. We solve this problem by postponing the check on the MACs (of opened values) to the output phase (of course, this may mean that some of the opened values are incorrect). During the output phase players generate a random linear combination of both the opened values and their shares of the corresponding MACs; they commit to the results and only then open α (see Figure 1). The intuition is that, because of the commitments, when α is revealed it is too late for corrupt players to exploit knowledge of the key. Therefore, if the MAC checks out, all opened values were correct with high probability, so we can trust that the output values we computed are correct and can safely open them.

Representation of values and MACs. In the online phase each shared value $a \in \mathbb{F}_{p^k}$ is represented as follows

$$\langle a \rangle := (\delta, (a_1, \ldots, a_n), (\gamma(a)_1, \ldots, \gamma(a)_n))$$

where $a = a_1 + \cdots + a_n$ and $\gamma(a)_1 + \cdots + \gamma(a)_n = \alpha(a + \delta)$. Player P_i holds $a_i, \gamma(a)_i$ and δ is public. The interpretation is that $\gamma(a) \leftarrow \gamma(a)_1 + \cdots + \gamma(a)_n$ is the MAC authenticating a under the global key α.

Computations. Using the natural component-wise addition of representations, and suppressing the underlying choices of $a_i, \gamma(a)_i$ for readability, we clearly have for secret values a, b and public constant e that

$$\langle a \rangle + \langle b \rangle = \langle a + b \rangle \quad e \cdot \langle a \rangle = \langle ea \rangle, \quad \text{and} \quad e + \langle a \rangle = \langle e + a \rangle,$$

where $e + \langle a \rangle := (\delta - e, (a_1 + e, a_2, \ldots, a_n), (\gamma(a)_1, \ldots, \gamma(a)_n))$. This possibility to easily add a public value is the reason for the "public modifier" δ in the definition of $\langle \cdot \rangle$. It is now clear that we can do secure linear computations directly on values represented this way.

What remains is multiplications: here we use the preprocessing. We would like the preprocessing to output random triples $\langle a \rangle, \langle b \rangle, \langle c \rangle$, where $c = ab$. However, our preprocessing produces triples which satisfy $c = ab + \Delta$, where Δ is an error that can be introduced by the adversary. We therefore need to check the triple before we use it. The check can be done by "sacrificing" another triple $\langle f \rangle, \langle g \rangle, \langle h \rangle$, where the same multiplicative equality should hold (see the protocol for details). Given such a valid triple, we can do multiplications in the following standard way: To compute $\langle xy \rangle$ we first open $\langle x \rangle - \langle a \rangle$ to get ϵ, and $\langle y \rangle - \langle b \rangle$ to get δ. Then $xy = (a + \epsilon)(b + \delta) = c + \epsilon b + \delta a + \epsilon\delta$. Thus, the new representation can be computed as

$$\langle x \rangle \cdot \langle y \rangle = \langle c \rangle + \epsilon \langle b \rangle + \delta \langle a \rangle + \epsilon\delta.$$

Protocol Π_{ONLINE}

Initialize: The parties first invoke the preprocessing to get the shared secret key $[\![\alpha]\!]$, a sufficient number of multiplication triples $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$, and pairs of random values $\langle r\rangle, [\![r]\!]$, as well as single random values $[\![t]\!], [\![e]\!]$. Then the steps below are performed in sequence according to the structure of the circuit to compute.

Input: To share P_i's input x_i, P_i takes an available pair $\langle r\rangle, [\![r]\!]$. Then, do the following:

1. $[\![r]\!]$ is opened to P_i (if it is known in advance that P_i will provide input, this step can be done already in the preprocessing stage).
2. P_i broadcasts $\epsilon \leftarrow x_i - r$.
3. The parties compute $\langle x_i\rangle \leftarrow \langle r\rangle + \epsilon$.

Add: To add two representations $\langle x\rangle, \langle y\rangle$,the parties locally compute $\langle x\rangle + \langle y\rangle$.

Multiply: To multiply $\langle x\rangle, \langle y\rangle$ the parties do the following:

1. They take two triples $(\langle a\rangle, \langle b\rangle, \langle c\rangle), (\langle f\rangle, \langle g\rangle, \langle h\rangle)$ from the set of the available ones and check that indeed $a \cdot b = c$.
   - Open a representation of a random value $[\![t]\!]$.
   - partially open $t \cdot \langle a\rangle - \langle f\rangle$ to get ρ and $\langle b\rangle - \langle g\rangle$ to get σ
   - evaluate $t \cdot \langle c\rangle - \langle h\rangle - \sigma \cdot \langle f\rangle - \rho \cdot \langle g\rangle - \sigma \cdot \rho$, and partially open the result.
   - If the result is not zero the players abort, otherwise go on with $\langle a\rangle, \langle b\rangle, \langle c\rangle$.

   Note that this check could in fact be done as part of the preprocessing. Moreover, it can be done for all triples in parallel, and so we actually need only one random value t.
2. The parties partially open $\langle x\rangle - \langle a\rangle$ to get ϵ and $\langle y\rangle - \langle b\rangle$ to get δ and compute $\langle z\rangle \leftarrow \langle c\rangle + \epsilon\langle b\rangle + \delta\langle a\rangle + \epsilon\delta$

Output: We enter this stage when the players have $\langle y\rangle$ for the output value y, but this value has been not been opened (the output value is only correct if players have behaved honestly). We then do the following:

1. Let $a_1, \ldots, a_T$ be all values publicly opened so far, where $\langle a_j\rangle = (\delta_j, (a_{j,1}, \ldots, a_{j,n}), (\gamma(a_j)_1, \ldots, \gamma(a_j)_n))$. Now, a random value $[\![e]\!]$ is opened, and players set $e_i = e^i$ for $i = 1, \ldots, T$. All players compute $a \leftarrow \sum_j e_j a_j$.
2. Each P_i calls $\mathcal{F}_{\text{COM}}$ to commit to $\gamma_i \leftarrow \sum_j e_j \gamma(a_j)_i$. For the output value $\langle y\rangle$, P_i also commits to his share y_i, and his share $\gamma(y)_i$ in the corresponding MAC.
3. $[\![\alpha]\!]$ is opened.
4. Each P_i asks $\mathcal{F}_{\text{COM}}$ to open γ_i, and all players check that $\alpha(a + \sum_j e_j\delta_j) = \sum_i \gamma_i$. If this is not OK, the protocol aborts. Otherwise the players conclude that the output value is correctly computed.
5. To get the output value y, the commitments to $y_i, \gamma(y)_i$ are opened. Now, y is defined as $y := \sum_i y_i$ and each player checks that $\alpha(y + \delta) = \sum_i \gamma(y)_i$, if so, y is the output.

Fig. 1. The online phase.

An important note is that during our protocol we are actually not guaranteed that we are working with the correct results, since we do not immediately check

the MACs of the opened values. During the first part of the protocol, parties will only do what we define as a *partial opening*, meaning that for a value $\langle a \rangle$, each party P_i sends a_i to P_1, who computes $a = a_1 + \cdots + a_n$ and broadcasts a to all players. We assume here for simplicity that we always go via P_1, whereas in practice, one would balance the workload over the players.

As sketched earlier we postpone the checking to the end of the protocol in the output phase. To check the MACs we need the global key α. We get α from the preprocessing but in a slightly different representation:

$$[\![\alpha]\!] := ((\alpha_1, \ldots, \alpha_n), (\beta_i, \gamma(\alpha)_1^i, \ldots, \gamma(\alpha)_n^i)_{i=1,\ldots,n}),$$

where $\alpha = \sum_i \alpha_i$ and $\sum_j \gamma(\alpha)_i^j = \alpha\beta_i$. Player P_i holds $\alpha_i, \beta_i, \gamma(\alpha)_1^i, \ldots, \gamma(\alpha)_n^i$. The idea is that $\gamma(\alpha)_i \leftarrow \sum_j \gamma(\alpha)_i^j$ is the MAC authenticating α under P_i's private key β_i. To open $[\![\alpha]\!]$ each P_j sends to each P_i his share α_j of α and his share $\gamma(\alpha)_i^j$ of the MAC on α made with P_i's private key and then P_i checks that $\sum_j \gamma(\alpha)_i^j = \alpha\beta_i$. (To open the value to only one party P_i, the other parties will simply send their shares only to P_i, who will do the checking. Only shares of α and $\alpha\beta_i$ are needed.)

Finally, the preprocessing will also output n pairs of a random value r in both of the presented representations $\langle r \rangle, [\![r]\!]$. These pairs are used in the Input phase of the protocol.

The full protocol for the online phase is shown in Figure 1. It assumes access to a commitment functionality $\mathcal{F}_{\text{COM}}$ that simply receives values to commit to from players, stores them and reveals a value to all players on request from the committer. Such a functionality could be implemented efficiently based, e.g., on Paillier encryption or the DDH assumption [8, 14]. However, we show in the full version that we can do ideal commitments based only on $\mathcal{F}_{\text{PREP}}$ and with cost $O(n^2)$ computation and communication.

Complexity. The (amortized) cost of a secure multiplication is easily seen to be $O(n)$ local elementary operations in $\mathbb{F}_{p^k}$, and communication of $O(n)$ field elements. Linear operations have the same computational cost but require no communication. The input stage requires $O(n)$ communication and computation to open $[\![r]\!]$ to P_i and one broadcast. Doing the output stage requires opening $O(n)$ commitments. In fact, the total number of commitments used is also $O(n)$, so this adds an $O(n^3)$ term to the complexity. In total, we therefore get the complexity claimed in the introduction: $O(n \cdot |C| + n^3)$ elementary field operations and storage/communication complexity $O(n \cdot |C| + n^3)$ field elements.

We can now state the theorem on security of the online phase, and its proof can be found in the full version.

Theorem 1. *In the $\mathcal{F}_{\text{PREP}}, \mathcal{F}_{\text{COM}}$-hybrid model, the protocol Π_{ONLINE} implements $\mathcal{F}_{\text{AMPC}}$ with statistical security against any static[4] active adversary corrupting up to $n-1$ parties.*

[4] The protocol is in fact adaptively secure, here we only show static security since our preprocessing is anyway only statically secure.

Based on a result from [18], we can also show a lower bound on the amount of preprocessing data and work required for a protocol. The proof is in the full version.

Theorem 2. *Assume a protocol π is the preprocessing model can compute any circuit over $\mathbb{F}_{p^k}$ of size at most S, with security against active corruption of at most $n-1$ players. We assume that the players supply roughly the same number of inputs ($O(S/n)$ each), and that any any player may receive output. Then the preprocessing must output $\Omega(S \log p^k)$ bits to each player, and for any player P_i, there exists a circuit C satisfying the conditions above, where secure computation of C requires P_i to execute an expected number of bit operations that is $\Omega(S \log p^k)$.*

It is easy to see that our protocol satisfies the conditions in the the theorem and that it meets the first bound up to a constant factor and the second up to a poly-logarithmic factor (as a function of the security parameter).

3 The Abstract Somewhat Homomorphic Encryption Scheme

In this section we specify the abstract properties we need for our cryptosystem. A concrete instantiation is found in Section 6.

We first define the plaintext space M. This will be given by a direct product of finite fields $(\mathbb{F}_{p^k})^s$ of characteristic p. Componentwise addition and multiplication of elements in M will be denoted by $+$ and $\cdot$. We assume there is an injective encoding function encode which takes elements in $(\mathbb{F}_{p^k})^s$ to elements in a ring R which is equal $\mathbb{Z}^N$ (as a $\mathbb{Z}$-module) for some integer N. We also assume a decode function which takes *arbitrary* elements in $\mathbb{Z}^N$ and returns an element in $(\mathbb{F}_{p^k})^s$. We require that for all $\mathbf{m} \in M$ that $\mathsf{decode}(\mathsf{encode}(\mathbf{m})) = \mathbf{m}$ and that the decode operation is compatible with the characteristic of the field, i.e. for any $\mathbf{x} \in \mathbb{Z}^N$ we have $\mathsf{decode}(\mathbf{x}) = \mathsf{decode}(\mathbf{x} \pmod p)$. And finally that the encoding function produces "short" vectors. More precisely, that for all $\mathbf{m} \in (\mathbb{F}_{p^k})^s$ $\|\mathsf{encode}(\mathbf{m})\|_\infty \leq \tau$ where $\tau = p/2$.

The two operations in R will be denoted by $+$ and $\cdot$. The addition operation in R is assumed to be componentwise addition, whereas we make no assumption on multiplication. All we require is that the following properties hold, for all elements $\mathbf{m}_1, \mathbf{m}_2 \in M$;

$$\mathsf{decode}(\mathsf{encode}(\mathbf{m}_1) + \mathsf{encode}(\mathbf{m}_2)) = \mathbf{m}_1 + \mathbf{m}_2,$$
$$\mathsf{decode}(\mathsf{encode}(\mathbf{m}_1) \cdot \mathsf{encode}(\mathbf{m}_2)) = \mathbf{m}_1 \cdot \mathbf{m}_2.$$

From now on, when we discuss the plaintext space M we assume it comes implicitly with the encode and decode functions for some integer N. If an element in M has the same component in each of the s-slots, then we call it a "diagonal" element. We let $\mathsf{Diag}(x)$ for $x \in \mathbb{F}_{p^k}$ denote the element $(x, x, \ldots, x) \in (\mathbb{F}_{p^k})^s$.

Our cryptosystem consists of a tuple $(\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{Enc}, \mathsf{Dec})$ of algorithms defined below, and parametrized by a security parameter κ.

$\mathsf{ParamGen}(1^\kappa, M)$: This parameter generation algorithm outputs an integer N (as above), definitions of the encode and decode functions, and a description of a randomized algorithm D_ρ^d, which outputs vectors in $\mathbb{Z}^d$. We assume that D_ρ^d outputs $\mathbf{r}$ with $\|\mathbf{r}\|_\infty \leq \rho$, except with negligible probability. The algorithm D_ρ^d is used by the encryption algorithm to select the random coins needed during encryption. The algorithm $\mathsf{ParamGen}$ also outputs an additive abelian group G. The group G also possesses a (not necessarily closed) multiplicative operator, which is commutative and distributes over the additive group of G. The group G is the group in which the ciphertexts will be assumed to lie. We write $\boxplus$ and $\boxtimes$ for the operations on G, and extend these in the natural way to vectors and matrices of elements of G. Finally $\mathsf{ParamGen}$ outputs a set C of allowable arithmetic SIMD circuits over $(\mathbb{F}_{p^k})^s$, these are the set of functions which our scheme will be able to evaluate ciphertexts over. We can think of C as a subset of $\mathbb{F}_{p^k}[X_1, X_2, \ldots, X_n]$, where we evaluate a function $f \in \mathbb{F}_{p^k}[X_1, X_2, \ldots, X_n]$ a total of s times in parallel on inputs from $(\mathbb{F}_{p^k})^n$. We assume that all other algorithms take as implicit input the output $P \leftarrow (1^\kappa, N, \mathsf{encode}, \mathsf{decode}, \mathsf{D}_\rho^d, G, C)$ of $\mathsf{ParamGen}$.
$\mathsf{KeyGen}()$: This algorithm outputs a public key pk and a secret key sk.
$\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}, \mathbf{r})$: On input of $\mathbf{x} \in \mathbb{Z}^N$, $\mathbf{r} \in \mathbb{Z}^d$, this deterministic algorithm outputs a ciphertext $c \in G$. When applying this algorithm one would obtain $\mathbf{x}$ from the application of the encode function, and $\mathbf{r}$ by calling D_ρ^d. This is what we mean when we write $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m})$, where $\mathbf{m} \in M$. However, it is convenient for us to define Enc on the intermediate state, $\mathbf{x} = \mathsf{encode}(\mathbf{m})$. To ease notation we write $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x})$ if the value of the randomness $\mathbf{r}$ is not important for our discussion. To make our zero-knowledge proofs below work, we will require that addition of V "clean" ciphertexts (for "small" values of V), of plaintext $\mathbf{x}_i$ in $\mathbb{Z}^N$, using randomness $\mathbf{r}_i$, results in a ciphertext which could be obtained by adding the plaintexts and randomness, as integer vectors, and then applying $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}, \mathbf{r})$, i.e.

$$\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_1 + \cdots + \mathbf{x}_V, \mathbf{r}_1 + \cdots + \mathbf{r}_V) = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_1, \mathbf{r}_1) \boxplus \cdots \boxplus \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_V, \mathbf{r}_V).$$

$\mathsf{Dec}_{\mathsf{sk}}(c)$: On input the secret key and a ciphertext c it returns either an element $\mathbf{m} \in M$, or the symbol $\perp$.

We are now able to define various properties of the above abstract scheme that we will require. But first a bit of notation: For a function $f \in C$ we let $n(f)$ denote the number of variables in f, and we let $\widehat{f}$ denote the function on G induced by f. That is, given f, we replace every $+$ operation with a $\boxplus$, every $\cdot$ operation is replaced with a $\boxtimes$ and every constant c is replaced by $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{encode}(c), \mathbf{0})$. Also, given a set of $n(f)$ vectors $\mathbf{x}_1, \ldots, \mathbf{x}_{n(f)}$, we define $f(\mathbf{x}_1, \ldots, \mathbf{x}_{n(f)})$ in the natural way by applying f in parallel on each coordinate.
Correctness: Intuitively correctness means that if one decrypts the result of a function $f \in C$ applied to $n(f)$ encrypted vectors of variables, then this should return the same value as applying the function to the $n(f)$ plaintexts. However, to apply the scheme in our protocol, we need to be a bit more liberal, namely the decryption result should be correct, even if the ciphertexts we start from were not necessarily generated by the normal encryption algorithm. They

only need to "contain" encodings and randomness that are not too large, such that the encodings decode to legal values. Formally, the scheme is said to be (B_{plain}, B_{rand}, C)-correct if

$$\begin{aligned}\Pr\,[\,&P \leftarrow \mathsf{ParamGen}(1^\kappa, M),\ \ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(),\ \ \text{for any } f \in C,\\ &\text{any } \mathbf{x}_i, \mathbf{r}_i,\ \text{with } \|\mathbf{x}_i\|_\infty \le B_{plain},\ \|\mathbf{r}_i\|_\infty \le B_{rand},\ \mathsf{decode}(\mathbf{x}_i) \in (\mathbb{F}_{p^k})^s,\\ &i = 1, \ldots, n(f),\ \text{and } c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_i, \mathbf{r}_i),\ c \leftarrow \widehat{f}(c_1, \ldots, c_{n(f)}) :\\ &\mathsf{Dec}_{\mathsf{sk}}(c) \neq f(\mathsf{decode}(\mathbf{x}_1), \ldots, \mathsf{decode}(\mathbf{x}_{n(f)}))\,] \qquad < \epsilon(\kappa).\end{aligned}$$

We will say that a ciphertext is (B_{plain}, B_{rand}, C)-*admissible* if it can be obtained as the ciphertext c in the above experiment, i.e., by applying a function from C to ciphertexts generated from (legal) encodings and randomness that are bounded by B_{plain} and B_{rand}.

$\mathsf{KeyGen}^*()$: This is a randomized algorithm that outputs a *meaningless public key* $\widetilde{\mathsf{pk}}$. We require that an encryption of any message $\mathsf{Enc}_{\widetilde{\mathsf{pk}}}(\mathbf{x})$ is statistically indistinguishable from an encryption of 0. Furthermore, if we set $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$ and $\widetilde{\mathsf{pk}} \leftarrow \mathsf{KeyGen}^*()$, then pk and $\widetilde{\mathsf{pk}}$ are computationally indistinguishable. This implies the scheme is IND-CPA secure in the usual sense.

Distributed Decryption: We assume, as a set up assumption, that a common public key has been set up where the secret key has been secret-shared among the players in such a way that they can collaborate to decrypt a ciphertext. We assume throughout that only (B_{plain}, B_{rand}, C)-admissible ciphertexts are to be decrypted, this constraint is guaranteed by our main protocol.

We note that some set-up assumption is always required to show UC security which is our goal here. Concretely, we assume that a functionality $\mathcal{F}_{\text{KeyGen}}$ is available, as specified in Figure 2. It basically generates a key pair and secret-shares the secret key among the players using a secret-sharing scheme that is assumed to be given as part of the specification of the cryptosystem. Since we want to allow corruption of all but one player, the maximal unqualified sets must be all sets of $n-1$ players.

Functionality $\mathcal{F}_{\text{KeyGen}}$

1. When receiving "start" from all honest players, run $P \leftarrow \mathsf{ParamGen}(1^\kappa, M)$, and then, using the parameters generated, run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$ (recall P, and hence 1^κ, is an implicit input to all functions we specify). Send pk to the adversary.
2. We assume a secret sharing scheme is given with which sk can be secret-shared. Receive from the adversary a set of shares s_j for each corrupted player P_j.
3. Construct a complete set of shares $(s_1, \ldots, s_n)$ consistent with the adversary's choices and sk. Note that this is always possible since the corrupted players form an unqualified set. Send pk to all players and s_i to each honest P_i.

Fig. 2. The Ideal Functionality for Distributed Key Generation

We note that it is possible to make a weaker set-up assumption, such as a common reference string (CRS), and using a general UC secure multiparty computation protocol for the CRS model to implement $\mathcal{F}_{\text{KeyGen}}$. While this may not be very efficient, one only needs to run this protocol once in the life-time of the system.

We also want our cryptosystem to implement the functionality $\mathcal{F}_{\text{KeyGenDec}}$ in Figure 3, which essentially specifies that players can cooperate to decrypt a (B_{plain}, B_{rand}, C)-admissible ciphertext, but the protocol is only secure against a passive attack: the adversary gets the correct decryption result, but can decide which result the honest players should learn.

Functionality $\mathcal{F}_{\text{KeyGenDec}}$

1. When receiving "start" from all honest players, run $\mathsf{ParamGen}(1^\kappa, M)$, and then, using the parameters generated, run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$. Send pk to the adversary and to all players, and store sk.
2. Hereafter on receiving "decrypt c" for (B_{plain}, B_{rand}, C)-admissible c from all honest players, send c and $m \leftarrow \mathsf{Dec}_{\mathsf{sk}}(c)$ to the adversary. On receiving m' from the adversary, send "Result m'" to all players, Both m and m' may be a special symbol $\perp$ indicating that decryption failed.
3. On receiving "decrypt c to P_j" for admissible c, if P_j is corrupt, send $c, m \leftarrow \mathsf{Dec}_{\mathsf{sk}}(c)$ to the adversary. If P_j is honest, send c to the adversary. On receiving δ from the adversary, if $\delta \notin M$, send $\perp$ to P_j, if $\delta \in M$, send $\mathsf{Dec}_{\mathsf{sk}}(c) + \delta$ to P_j.

Fig. 3. The Ideal Functionality for Distributed Key Generation and Decryption

We are now finally ready to define the basic set of properties that the underlying cryptosystem should satisfy, in order to be used in our protocol. Here we use an "information theoretic" security parameter sec that controls the errors in our ZK proofs below.

Definition 1. (Admissible Cryptosystem.) *Let C contain formulas of form $(x_1+\cdots+x_n)\cdot(y_1+\cdots+y_n)+z_1+\cdots+z_n$, as well as all "smaller" formulas , i.e., with a smaller number of additions and possibly no multiplication. A cryptosystem is* admissible *if it is defined by algorithms* $(\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{Enc}, \mathsf{Dec})$ *with properties as defined above, is (B_{plain}, B_{rand}, C)-correct, where*

$$B_{plain} = N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{(1/2+\nu)\mathsf{sec}}, \quad B_{rand} = d \cdot \rho \cdot \mathsf{sec}^2 \cdot 2^{(1/2+\nu)\mathsf{sec}};$$

and where $\nu > 0$ can be an arbitrary constant. Finally there exist a secret sharing scheme as required in $\mathcal{F}_{\text{KeyGen}}$ and a protocol $\Pi_{\mathsf{KeyGenDec}}$ with the property that when composed with $\mathcal{F}_{\text{KeyGen}}$ it securely implements the functionality $\mathcal{F}_{\text{KeyGenDec}}$.

The set C is defined to contain all computations on ciphertext that we need in our main protocol. Throughout the paper we will assume that B_{plain}, B_{rand} are defined as here in terms of τ, ρ and sec. This is because these are the bounds we can force corrupt players to respect via our zero-knowledge protocol, as we shall see.

4 Zero-Knowledge Proof of Plaintext Knowledge

This section presents a zero-knowledge protocol that takes as input sec ciphertexts $c_1, \ldots, c_{\mathsf{sec}}$ generated by one of the players in our protocol, who will act as the prover. If the prover is honest then $c_i = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_i, \mathbf{r}_i)$, where $\mathbf{x}_i$ has been obtained from the encode function, i.e. $\|\mathbf{x}_i\|_\infty \leq \tau$, and $\mathbf{r}_i$ has been generated from D_ρ^d (so we may assume that $\|\mathbf{r}_i\|_\infty \leq \rho$). Our protocol is a zero-knowledge proof of plaintext knowledge (ZKPoPK) for the following relation:

$$\begin{aligned} R_{\mathsf{PoPK}} = \{\ (x,w) |\ & x = (\mathsf{pk}, \mathbf{c}),\ w = ((\mathbf{x}_1, \mathbf{r}_1), \ldots, (\mathbf{x}_{\mathsf{sec}}, \mathbf{r}_{\mathsf{sec}})) : \\ & \mathbf{c} = (c_1, \ldots, c_{\mathsf{sec}}),\ c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_i, \mathbf{r}_i), \\ & \|\mathbf{x}_i\|_\infty \leq B_{plain},\ \mathsf{decode}(\mathbf{x}_i) \in (\mathbb{F}_{p^k})^s,\ \|\mathbf{r}_i\|_\infty \leq B_{rand}\ \}\,. \end{aligned}$$

The zero-knowledge and completeness properties hold only if the ciphertexts c_i satisfy $\|\mathbf{x}_i\|_\infty \leq \tau$ and $\|\mathbf{r}_i\|_\infty \leq \rho$.

In our preprocessing protocol, players will be required to give such a ZKPoPK for all ciphertexts they provide. By admissibility of the cryptosystem, this will imply that every ciphertext occurring in the protocol will be (B_{plain}, B_{rand}, C)-admissible and can therefore be decrypted correctly. The ZKPoPK can also be called with a flag diag which will modify the proof so that it additionally proves that $\mathsf{decode}(\mathbf{x}_i)$ is a diagonal element.

The protocol is not meant to implement an ideal functionality, but we can still use it and prove UC security for the main protocol, since we will always generate the challenge $\mathbf{e}$ by calling the $\mathcal{F}_{\text{RAND}}$ ideal functionality (see the full version for more details).

The protocol and its proof of security are given in the full version and its computational complexity per ciphertext is essentially the cost of a constant number of encryptions. In the full version, we also give a variant of the ZK proof that allows even smaller values for B_{plain}, B_{rand}, namely $B_{plain} = N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{\mathsf{sec}/2+8}$, $B_{rand} = d \cdot \rho \cdot \mathsf{sec}^2 \cdot 2^{\mathsf{sec}/2+8}$, and hence improves performance further. This variant is most efficient when executed using the Fiat-Shamir heuristic (although it can also work without random oracles), and we believe this variant is the best for a practical implementation.

5 The Preprocessing Phase

In this section we construct the protocol Π_{PREP} which securely implements the functionality $\mathcal{F}_{\text{PREP}}$ in the presence of functionalities $\mathcal{F}_{\text{KeyGenDec}}$ (Figure 3) and $\mathcal{F}_{\text{RAND}}$. The preprocessing uses the above abstract cryptosystem with $M = (\mathbb{F}_{p^k})^s$, but the online phase is designed for messages in $\mathbb{F}_{p^k}$. Therefore, we extend the notation $\langle\cdot\rangle$ and $[\![\cdot]\!]$ to messages in M: since addition and multiplication on M are componentwise, for $\mathbf{m} = (m_1, \ldots, m_s)$, we define $\langle\mathbf{m}\rangle = (\langle m_1\rangle, \ldots, \langle m_s\rangle)$ and similarly for $[\![\mathbf{m}]\!]$. Conversely, once a representation (or a pair, triple) on vectors is produced in the preprocessing, it will be disassembled into its coordinates, so that it can be used in the online phase. In Figures 4,5 and 6, we

introduce subprotocols that are accessed by the main preprocessing protocol in several steps. Note that the subprotocols are not meant to implement ideal functionalities: their purpose is merely to summarize parts of the main protocol that are repeated in various occasions. Theorem 3 below is proved in the full version.

Theorem 3. *The protocol* Π_{PREP} *(Figure 7) implements* $\mathcal{F}_{\text{PREP}}$ *with computational security against any static, active adversary corrupting up to* $n-1$ *parties, in the* $\mathcal{F}_{\text{KeyGen}}, \mathcal{F}_{\text{Rand}}$*-hybrid model when the underlying cryptosystem is admissible*[5]*.*

Protocol Reshare

Usage: Input is $e_{\mathbf{m}}$, where $e_{\mathbf{m}} = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m})$ is a public ciphertext and a parameter *enc*, where *enc* = NewCiphertext or *enc* = NoNewCiphertext. Output is a share $\mathbf{m}_i$ of $\mathbf{m}$ to each player P_i; and if *enc* = NewCiphertext, a ciphertext $e'_{\mathbf{m}}$. The idea is that $e_{\mathbf{m}}$ could be a product of two ciphertexts, which Reshare converts to a "fresh" ciphertext $e'_{\mathbf{m}}$. Since Reshare uses distributed decryption (that may return an incorrect result), it is not guaranteed that $e_{\mathbf{m}}$ and $e'_{\mathbf{m}}$ contain the same value, but it *is* guaranteed that $\sum_i \mathbf{m}_i$ is the value contained in $e'_{\mathbf{m}}$.

Reshare$(e_{\mathbf{m}}, enc)$:

1. Each P_i samples a uniform $\mathbf{f}_i \in (\mathbb{F}_{p^k})^s$. Define $\mathbf{f} := \sum_{i=1}^{n} \mathbf{f}_i$.
2. Each P_i computes and broadcasts $e_{\mathbf{f}_i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{f}_i)$.
3. Each P_i runs Π_{ZKPoPK} as a prover on $e_{\mathbf{f}_i}$. The protocol aborts if any proof fails.
4. The players compute $e_{\mathbf{f}} \leftarrow e_{\mathbf{f}_1} \boxplus \cdots \boxplus e_{\mathbf{f}_n}$, and $e_{\mathbf{m}+\mathbf{f}} \leftarrow e_{\mathbf{m}} \boxplus e_{\mathbf{f}}$.
5. The players invoke $\mathcal{F}_{\text{KeyGenDec}}$ to decrypt $e_{\mathbf{m}+\mathbf{f}}$ and thereby obtain $\mathbf{m}+\mathbf{f}$.
6. P_1 sets $\mathbf{m}_1 \leftarrow \mathbf{m}+\mathbf{f}-\mathbf{f}_1$, and each player P_i $(i \neq 1)$ sets $\mathbf{m}_i \leftarrow -\mathbf{f}_i$.
7. If *enc* = NewCiphertext, all players set $e'_{\mathbf{m}} \leftarrow \mathsf{Enc}_{pk}(\mathbf{m}+\mathbf{f}) \boxminus e_{\mathbf{f}_1} \boxminus \cdots \boxminus e_{\mathbf{f}_n}$, where a default value for the randomness is used when computing $\mathsf{Enc}_{pk}(\mathbf{m}+\mathbf{f})$.

Fig. 4. The sub-protocol for additively secret sharing a plaintext $\mathbf{m} \in (\mathbb{F}_{p^k})^s$ on input a ciphertext $e_{\mathbf{m}} = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m})$.

6 Concrete Instantiation of the Abstract Scheme based on LWE

We now describe the concrete scheme, which is based on the somewhat homomorphic encryption scheme of Brakerski and Vaikuntanathan (BV) [5]. The main differences are that we are only interested in evaluation of circuits of multiplicative depth one, we are interested in performing operations in parallel on multiple data items, and we require a distributed decryption procedure. In this section we detail the scheme and the distributed decryption procedure; in the full version we discuss security of the scheme, and present some sample parameter sizes and performance figures.

[5] The definition of admissible cryptosystem demands a decryption protocol that implements $\mathcal{F}_{\text{KeyGenDec}}$ based on $\mathcal{F}_{\text{KeyGen}}$, hence the theorem only assumes $\mathcal{F}_{\text{KeyGen}}$.

Protocol $\mathsf{PBracket}$

Usage: On input shares $\mathbf{v}_1, \ldots, \mathbf{v}_n$ privately held by the players and public ciphertext $e_{\mathbf{v}}$, this protocol generates $[\![\mathbf{v}]\!]$. It is assumed that $\sum_i \mathbf{v}_i$ is the plaintext contained in $e_{\mathbf{v}}$.

$\mathsf{PBracket}(\mathbf{v}_1, \ldots, \mathbf{v}_n, e_{\mathbf{v}})$:

1. For $i = 1, \ldots, n$
   (a) All players set $e_{\gamma_i} \leftarrow e_{\beta_i} \boxtimes e_{\mathbf{v}}$ (note that e_{β_i} is generated during the initialization process, and known by every player)
   (b) Players generate $(\gamma_i^1, \ldots \gamma_i^n) \leftarrow \mathsf{Reshare}(e_{\gamma_i}, \mathsf{NoNewCiphertext})$, so each player P_j gets a share γ_i^j of $\mathbf{v} \cdot \beta_i$.
2. Output the representation $[\![\mathbf{v}]\!] = (\mathbf{v}_1, \ldots, \mathbf{v}_n, (\beta_i, \gamma_1^i, \ldots, \gamma_n^i)_{i=1,\ldots,n})$.

Fig. 5. The sub-protocol for generating $[\![\mathbf{v}]\!]$.

Protocol PAngle

Usage: On input shares $\mathbf{v}_1, \ldots, \mathbf{v}_n$ privately held by the players and public ciphertext $e_{\mathbf{v}}$, this protocol generates $\langle \mathbf{v} \rangle$. It is assumed that $\sum_i \mathbf{v}_i$ is the plaintext contained in $e_{\mathbf{v}}$.

$\mathsf{PAngle}(\mathbf{v}_1, \ldots, \mathbf{v}_n, e_{\mathbf{v}})$:

1. All players set $e_{\mathbf{v}\cdot\alpha} \leftarrow e_{\mathbf{v}} \boxtimes e_{\alpha}$ (note that e_{α} is generated during the initialization process, and known by every player)
2. Players generate $(\gamma_1, \ldots, \gamma_n) \leftarrow \mathsf{Reshare}(e_{\mathbf{v}\cdot\alpha}, \mathsf{NoNewCiphertext})$, so each player P_i gets a share γ_i of $\alpha \cdot \mathbf{v}$.
3. Output representation $\langle \mathbf{v} \rangle = (0, \mathbf{v}_1, \ldots, \mathbf{v}_n, \gamma_1, \ldots, \gamma_n)$.

Fig. 6. The sub-protocol for generating $\langle \mathbf{v} \rangle$.

$\mathsf{ParamGen}(1^\kappa, M)$: Recall the message space is given by $M = (\mathbb{F}_{p^k})^s$ for two integers k and s, and a prime p, i.e. the message space is s copies of the finite field $\mathbb{F}_{p^k}$. To map this to our scheme below, one first finds a cyclotomic polynomial $F(X) := \Phi_m(X)$ of degree $N := \phi(m)$, where N is lower bounded by some function of the security parameter κ. The polynomial $F(X)$ needs to be such that modulo p the polynomial $F(X)$ factors into l' irreducible factors of degree k' where $l' \geq s$ and k divides k'. We then define an algebra A_p as $A_p := \mathbb{F}_p[X]/F(X)$ and we have an embedding of M into A_p, $\phi : M \to A_p$. By "lifting" modulo p we see that there is a natural inclusion $\iota : A_p \to \mathbb{Z}^N$, which maps the polynomial of degree less than N with coefficients in $\mathbb{F}_p$ into the integer vector of length N with coefficients in the range $(-p/2, \ldots, p/2]$. The encode function is then defined by $\iota(\phi(\mathbf{m}))$ for $\mathbf{m} \in (\mathbb{F}_{p^k})^s$, with decode defined by $\phi^{-1}(\mathbf{x} \pmod p)$ for $\mathbf{x} \in \mathbb{Z}^N$. It is clear, by choice of the natural inclusion ι, that $\|\mathsf{encode}(\mathbf{m})\|_\infty \leq p/2 = \tau$.

We pick a large integer q, whose size we will determine later, and defined $A_q := (\mathbb{Z}/q\mathbb{Z})[X]/F(X)$, i.e. the ring of integer polynomials modulo reduction by $F(X)$ and q. In practice we consider the image of encode to lie in A_q, and thus we abuse notation, by writing addition and multiplication in A_q by $+$ and $\cdot$. Note, that this means that applying decode to elements obtained from encode followed by a series of arithmetic operations may not result in the value in M which one would expect. This corresponds to where our scheme can only evaluate circuits from a given set C.

Protocol Π_{PREP}

Usage: The Triple-step is always executed sec times in parallel. This ensures that when calling Π_{ZKPoPK}, we can always give it the sec ciphertexts it requires as input. In addition both Π_{ZKPoPK} and Π_{PREP} can be executed in a SIMD fashion, i.e. they are data-oblivious bar when they detect an error. Thus we can execute Π_{ZKPoPK} and Π_{PREP} on the packed plaintext space $(\mathbb{F}_{p^k})^s$. Thereby, we generate $s \cdot \mathsf{sec}$ elements in one go and then buffer the generated triples, outputting the next unused one on demand.

Initialize: This step generates the global key α and "personal keys" β_i.

1. The players call "start" on $\mathcal{F}_{\text{KeyGenDec}}$ to obtain the public key pk
2. Each P_i generates a MAC-key $\beta_i \in \mathbb{F}_{p^k}$
3. Each P_i generates $\alpha_i \in \mathbb{F}_{p^k}$. Let $\alpha := \sum_{i=1}^{n} \alpha_i$
4. Each P_i computes and broadcasts $e_{\alpha_i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathsf{Diag}(\alpha_i))$, $e_{\beta_i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathsf{Diag}(\beta_i))$
5. Each P_i invokes Π_{ZKPoPK} (with diag set to true) as prover on input $(e_{\alpha_i}, \ldots, e_{\alpha_i})$ and on input $(e_{\beta_i}, \ldots, e_{\beta_i})$, where $e_{\alpha_i}, e_{\beta_i}$ are repeated sec times, which is the number of ciphertexts Π_{ZKPoPK} requires as input. (This is not very efficient, but only needs to be done once for each player.)
6. All players compute $e_\alpha \leftarrow e_{\alpha_1} \boxplus \cdots \boxplus e_{\alpha_n}$, and generate $[\![\mathsf{Diag}(\alpha)]\!] \leftarrow \mathsf{PBracket}(\mathsf{Diag}(\alpha_1), \ldots, \mathsf{Diag}(\alpha_n), e_\alpha)$

Pair: This step generates a pair $[\![\mathbf{r}]\!], \langle \mathbf{r} \rangle$, and can be used to generate a single value $[\![\mathbf{r}]\!]$, by not performing the call to Pangle

1. Each P_i generates $\mathbf{r}_i \in (\mathbb{F}_{p^k})^s$. Let $\mathbf{r} := \sum_{i=1}^{n} \mathbf{r}_i$
2. Each P_i computes and broadcasts $e_{\mathbf{r}_i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{r}_i)$. Let $e_{\mathbf{r}} = e_{\mathbf{r}_1} \boxplus \cdots \boxplus e_{\mathbf{r}_n}$
3. Each P_i invokes Π_{ZKPoPK} as prover on the ciphertext he generated
4. Players generate $[\![\mathbf{r}]\!] \leftarrow \mathsf{PBracket}(\mathbf{r}_1, \ldots, \mathbf{r}_n, e_{\mathbf{r}})$, $\langle \mathbf{r} \rangle \leftarrow \mathsf{PAngle}(\mathbf{r}_1, \ldots, \mathbf{r}_n, e_{\mathbf{r}})$

Triple: This step generates a multiplicative triple $\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle, \langle \mathbf{c} \rangle$

1. Each P_i generates $\mathbf{a}_i, \mathbf{b}_i \in (\mathbb{F}_{p^k})^s$. Let $\mathbf{a} := \sum_{i=1}^{n} \mathbf{a}_i$, $\mathbf{b} := \sum_{i=1}^{n} \mathbf{b}_i$
2. Each P_i computes and broadcasts $e_{\mathbf{a}_i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{a}_i)$, $e_{\mathbf{b}_i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{b}_i)$
3. Each P_i invokes Π_{ZKPoPK} as prover on the ciphertexts he generated.
4. The players set $e_{\mathbf{a}} \leftarrow e_{\mathbf{a}_1} \boxplus \cdots \boxplus e_{\mathbf{a}_n}$ and $e_{\mathbf{b}} \leftarrow e_{\mathbf{b}_1} \boxplus \cdots \boxplus e_{\mathbf{b}_n}$
5. Players generate $\langle \mathbf{a} \rangle \leftarrow \mathsf{PAngle}(\mathbf{a}_1, \ldots, \mathbf{a}_n, e_{\mathbf{a}})$, $\langle \mathbf{b} \rangle \leftarrow \mathsf{PAngle}(\mathbf{b}_1, \ldots, \mathbf{b}_n, e_{\mathbf{b}})$.
6. All players compute $e_{\mathbf{c}} \leftarrow e_{\mathbf{a}} \boxtimes e_{\mathbf{b}}$
7. Players set $(\mathbf{c}_1, \ldots, \mathbf{c}_n, e'_{\mathbf{c}}) \leftarrow \mathsf{Reshare}(e_{\mathbf{c}}, \mathsf{NewCiphertext})$.
8. Players generate $\langle \mathbf{c} \rangle \leftarrow \mathsf{PAngle}(\mathbf{c}_1, \ldots, \mathbf{c}_n, e'_{\mathbf{c}})$.

Fig. 7. The protocol for constructing the global key $[\![\alpha]\!]$, pairs $[\![\mathbf{r}]\!], \langle \mathbf{r} \rangle$ and multiplicative triples $\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle, \langle \mathbf{c} \rangle$.

The ciphertext space G is defined to be A_q^3, with addition $\boxplus$ defined componentwise. The multiplicative operator $\boxtimes$ is defined as follows

$$(\mathbf{a}_0, \mathbf{a}_1, 0) \boxtimes (\mathbf{b}_0, \mathbf{b}_1, 0) := (\mathbf{a}_0 \cdot \mathbf{b}_0, \mathbf{a}_1 \cdot \mathbf{b}_0 + \mathbf{a}_0 \cdot \mathbf{b}_1, -\mathbf{a}_1 \cdot \mathbf{b}_1),$$

i.e. multiplication is only defined on elements whose third coefficient is zero.

We define D_ρ^d as follows: The discrete Gaussian $D_{\mathbb{Z}^N, s}$, with *Gaussian parameter* s, is defined to be the random variable on $\mathbb{Z}_q^N$ (centered around the origin)

obtained from sampling $\mathbf{x} \in \mathbb{R}^N$, with probability proportional to $\exp(-\pi \cdot \|\mathbf{x}\|_2/s^2)$, and then rounding the result to the nearest lattice point and reducing it modulo q. Note, sampling from the distribution with probability density function proportional to $\exp(-\pi \cdot \|\mathbf{x}\|_2/s^2)$, means using a normal variate with mean zero, and standard deviation $r := s/\sqrt{2 \cdot \pi}$. In our concrete scheme we set $d := 3 \cdot N$ and define D_ρ^d to be the distribution defined by $(D_{\mathbb{Z}^N,s})^3$. Note, that in the notation D_ρ^d the implicit dependence on q has been suppressed to ease readability. The determining of q and r as functions of all the other parameters, we leave until we discuss security of the scheme.

$\underline{\mathsf{KeyGen}()}$: We will use the public key version of the Brakerski–Vaikuntanathan scheme [5]. Given the above set up, key generation proceeds as follows: First one samples elements $\mathbf{a} \leftarrow A_q$ and $\mathbf{s}, \mathbf{e} \leftarrow D_{\mathbb{Z}^N,s}$. Then treating $\mathbf{s}$ and $\mathbf{e}$ as elements of A_q one computes $\mathbf{b} \leftarrow (\mathbf{a} \cdot \mathbf{s}) + (p \cdot \mathbf{e})$. The public and private key are then set to be $\mathsf{pk} \leftarrow (\mathbf{a}, \mathbf{b})$ and $\mathsf{sk} \leftarrow \mathbf{s}$.
$\underline{\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}, \mathbf{r})}$: Given a message $\mathbf{x} \leftarrow \mathsf{encode}(m)$ where $m \in M$, and $\mathbf{r} \in \mathsf{D}_\rho^d$, we proceed as follows: The element $\mathbf{r}$ is parsed as $(\mathbf{u}, \mathbf{v}, \mathbf{w}) \in (\mathbb{Z}^N)^3$. Then the encryptor computes $\mathbf{c}_0 \leftarrow (\mathbf{b} \cdot \mathbf{v}) + (p \cdot \mathbf{w}) + \mathbf{x}$ and $\mathbf{c}_1 \leftarrow (\mathbf{a} \cdot \mathbf{v}) + (p \cdot \mathbf{u})$. Finally returning the ciphertext $(\mathbf{c}_0, \mathbf{c}_1, 0)$.
$\underline{\mathsf{Dec}_{\mathsf{sk}}(c)}$: Given a secret key $\mathsf{sk} = \mathbf{s}$ and a ciphertext $c = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2)$ this algorithm computes the element in A_q satisfying $\mathbf{t} = \mathbf{c}_0 - (\mathbf{s} \cdot \mathbf{c}_1) - (\mathbf{s} \cdot \mathbf{s} \cdot \mathbf{c}_2)$. On reduction by q the value of $\|\mathbf{t}\|_\infty$ will be bounded by a relatively small constant B; assuming of course that the "noise" within a ciphertext has not grown too large. We shall refer to the value $\mathbf{t} \mod q$ as the "noise", despite it also containing the message to be decrypted. At this point the decryptor simply reduces $\mathbf{t}$ modulo p to obtain the desired plaintext in A_q, which can then be decoded via the decode algorithm.
$\underline{\mathsf{KeyGen}^*()}$: This simply samples $\widehat{\mathbf{a}}, \widehat{\mathbf{b}} \leftarrow A_q$ and returns $\widehat{\mathsf{pk}} := (\widehat{\mathbf{a}}, \widehat{\mathbf{b}})$.

Following the discussion in [5] we see that with this *fixed* ciphertext space, our scheme is somewhat homomorphic. It can support a relatively large number of addition operations, and a single multiplication.

Distributed Version We now extend the scheme above to enable distributed decryption. We first set up the distributed keys as follows. After invoking the functionality for key generation, each player obtains a share $\mathsf{sk}_i = (\mathbf{s}_{i,1}, \mathbf{s}_{i,2})$, these are chosen uniformly such that the master secret is written as

$$\mathbf{s} = \mathbf{s}_{1,1} + \cdots + \mathbf{s}_{n,1}, \quad \mathbf{s} \cdot \mathbf{s} = \mathbf{s}_{1,2} + \cdots + \mathbf{s}_{n,2}.$$

As remarked earlier this one-time setup procedure can be accomplished by standard UC-secure multiparty computation protocols such as that described in [3]. The following theorem is proved in the full version. It depends on the constant B defined above. In the full version we compute the value of B when the input ciphertext is (B_{plain}, B_{rand}, C)-admissible, and show how to choose parameters for the cryptosystem such that the required bound on B is satisfied.

Theorem 4. *In the $\mathcal{F}_{\text{KeyGen}}$-hybrid model, the protocol Π_{DDec} (Figure 8) implements $\mathcal{F}_{\text{KeyGenDec}}$ with statistical security against any static active adversary corrupting up to $n-1$ parties if $B + 2^{\mathsf{sec}} \cdot B < q/2$.*

Protocol Π_{DDec}

Initialize: Each party P_i on being given the ciphertext $c = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2)$, and an upper bound B on the infinity norm of $\mathbf{t}$ above, computes

$$\mathbf{v}_i \leftarrow \begin{cases} \mathbf{c}_0 - (\mathbf{s}_{i,1} \cdot \mathbf{c}_1) - (\mathbf{s}_{i,2} \cdot \mathbf{c}_2) & \text{if } i = 1 \\ -(\mathbf{s}_{i,1} \cdot \mathbf{c}_1) - (\mathbf{s}_{i,2} \cdot \mathbf{c}_2) & \text{if } i \neq 1 \end{cases}$$

and sets $\mathbf{t}_i \leftarrow \mathbf{v}_i + p \cdot \mathbf{r}_i$ where $\mathbf{r}_i$ is a random element with infinity norm bounded by $2^{\mathsf{sec}} \cdot B/(n \cdot p)$.

Public Decryption: All the players are supposed to learn the message.

- Each party P_i broadcasts $\mathbf{t}_i$
- All players compute $\mathbf{t}' \leftarrow \mathbf{t}_1 + \cdots + \mathbf{t}_n$ and obtain a message $m' \leftarrow \mathsf{decode}(\mathbf{t}' \bmod p)$.

Private Decryption: Only player P_j is supposed to learn the message.

- Each party P_i sends $\mathbf{t}_i$ to P_j
- P_j computes $\mathbf{t}' \leftarrow \mathbf{t}_1 + \cdots + \mathbf{t}_n$ and obtain a message $m' \leftarrow \mathsf{decode}(\mathbf{t}' \bmod p)$.

Fig. 8. The distributed decryption protocol.

7 Acknowledgements

The first, second and fourth author acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within which [part of] this work was performed; and also from the CFEM research center (supported by the Danish Strategic Research Council) within which part of this work was performed.

The third author was supported by the European Commission through the ICT Programme under Contract ICT-2007-216676 ECRYPT II and via an ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO, by EPSRC via grant COED–EP/I03126X, the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under agreement number FA8750-11-2-0079, and by a Royal Society Wolfson Merit Award. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, AFRL, the U.S. Government, the European Commission or EPSRC.

The authors would like to thank Robin Chapman, Henri Cohen and Rob Harley for various discussions whilst this work was carried out.

References

1. G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2012.
2. D. Beaver. Efficient multiparty protocols using circuit randomization. In J. Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.
3. R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, pages 169–188, 2011.
4. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:111, 2011.
5. Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In P. Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.
6. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.
7. I. Damgård, M. Keller, E. Larraia, C. Miles, and N. P. Smart. Implementing aes via an actively/covertly secure dishonest-majority mpc protocol. *IACR Cryptology ePrint Archive*, 2012:262, 2012.
8. I. Damgård and J. B. Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In M. Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 581–596. Springer, 2002.
9. I. Damgård and C. Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In *CRYPTO*, pages 558–576, 2010.
10. C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.
11. C. Gentry, S. Halevi, and N. P. Smart. Fully homomorphic encryption with polylog overhead. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012.
12. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In D. S. Johnson and U. Feige, editors, *STOC*, pages 21–30. ACM, 2007.
13. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In D. Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591. Springer, 2008.
14. Y. Lindell. Highly-efficient universally-composable commitments based on the ddh assumption. In *EUROCRYPT*, pages 446–466, 2011.
15. S. Myers, M. Sergi, and abhi shelat. Threshold fully homomorphic encryption and secure computation. *IACR Cryptology ePrint Archive*, 2011:454, 2011.
16. J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. *IACR Cryptology ePrint Archive*, 2011:91, 2011.
17. N. P. Smart and F. Vercauteren. Fully homomorphic simd operations. *IACR Cryptology ePrint Archive*, 2011:133, 2011.
18. S. Winkler and J. Wullschleger. On the efficiency of classical and quantum oblivious transfer reductions. In *CRYPTO*, pages 707–723, 2010.