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 [8] 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. [5, 13, 25]. The "MPC-in-the-head" technique of Ishai et al. [18, 17] 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 [15]. 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) [6]. 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].

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 Appendix A.4 for details.

Obtaining our result requires new ideas compared to [5], 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

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

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 [3], 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 [27] to handle many values in parallel in a single ciphertext.

To capitalize on this idea, we apply the SIMD approach to the cryptosystem from [7] (see also [16] 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 check involving the MACs. Finally we adapt a zero-knowledge proof of plaintext knowledge from [5] 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 [5, 13] 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 [7] 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 [5]. 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 [11].

Concurrent Related Work. In recent independent work [24, 2, 16], 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 [16], 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 [25], Nielsen et al. consider secure computing for Boolean Circuits. Their online phase is similar to that of [5], 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 [25] 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 Figure 15 in Appendix Ethe full version. 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 $\mathcal{F}_{\text{PREP}}$ (Figure 16 of Appendix E). 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 Appendix A.3 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 [5], 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 [5], and we can therefore do secure multiplication based on multiplication triples à la Beaver [3] 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.

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.

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

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

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 [12, 19]. However, we show in Appendix A.3 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 is in Appendix A.3.

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.*

Based on a result from [28], we can also show a lower bound on the amount of preprocessing data and work required for a protocol. The proof is in Appendix B.

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 $\mathsf{decode}(\mathbf{x}) = \mathsf{decode}(\mathbf{x}$

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

(mod 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

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

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 $\|\mathbf{r}_i\|_\infty \le \rho$). Our protocol is a zero-knowledge proof of plaintext knowledge (ZKPoPK) for the following relation:

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

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

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 Appendix E). Hence the honest-verifier ZK property implies straight-line simulation[5]. As for knowledge extraction, the UC simulator we construct in our security proof will know the secret key for the cryptosystem and can therefore extract a dishonest prover's witness simply by decrypting. In the reduction to show that the simulator works, we do not know the secret key, but here we are allowed to do extraction by rewinding.

The protocol and its proof of security are given in Appendix A.1, Figure 9 and its computational complexity per ciphertext is essentially the cost of a constant number of encryptions. In Appendix A.1, 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 $\mathcal{F}_{\text{PREP}}$ (specified in Figure 16) in the presence of functionalities $\mathcal{F}_{\text{KeyGenDec}}$ (Figure 3) and $\mathcal{F}_{\text{Rand}}$ (Figure 14). 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 Appendix A.5.

[5] $\mathcal{F}_{\text{Rand}}$ can be implemented by standard methods, and the complexity of this is not significant for the main protocol since we may use the same challenge for many instances of the proof, and each proof handles sec ciphertexts.

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*[6].

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 player 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 player P_i computes and broadcasts $e_{\mathbf{f}_i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{f}_i)$.
3. Each player P_i runs Π_{ZKPoPK} acting 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 $e_{\mathbf{m}} = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m})$.

Protocol 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 $e_{\mathbf{v}}$.

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 $(\gamma_i^1, \ldots \gamma_i^n) \leftarrow$ Reshare$(e_{\gamma_i}$, 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 $e_{\mathbf{v}}$.

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 $(\gamma_1, \ldots, \gamma_n) \leftarrow$ Reshare$(e_{\mathbf{v}\cdot\alpha}$, 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 $\langle \mathbf{v} \rangle$.

[6] 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 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 player P_i generates a MAC-key $\beta_i \in \mathbb{F}_{p^k}$
3. Each player P_i generates $\alpha_i \in \mathbb{F}_{p^k}$. Let $\alpha := \sum_{i=1}^{n} \alpha_i$
4. Each player 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 player P_i invokes Π_{ZKPoPK} (with diag set to true) acting 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 player P_i generates $\mathbf{r}_i \in (\mathbb{F}_{p^k})^s$. Let $\mathbf{r} := \sum_{i=1}^{n} \mathbf{r}_i$
2. Each player 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 player P_i invokes Π_{ZKPoPK} acting 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 player 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 player 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 player P_i invokes Π_{ZKPoPK} acting 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 $\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle, \langle \mathbf{c} \rangle$.

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) [7]. 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 Appendix D we discuss security of the scheme, and present some sample parameter sizes and performance figures.

$\underline{\mathsf{ParamGen}}(1^\kappa, M)$: 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.

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 [7]. 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 [7] 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 [5]. The following theorem is proved in Appendix A.6. It depends on the constant B defined above. In Appendix D 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. S. Arora and R. Ge. New algorithms for learning in presence of errors. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP (1)*, volume 6755 of *Lecture Notes in Computer Science*, pages 403–415. Springer, 2011.
2. 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.
3. 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.
4. E. Ben-Sasson, S. Fehr, and R. Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. *IACR Cryptology ePrint Archive*, 2011:629, 2011.
5. R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, pages 169–188, 2011.
6. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:111, 2011.
7. 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.
8. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.
9. Y. Chen and P. Q. Nguyen. Bkz 2.0: Better lattice security estimates. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.
10. R. Cramer, I. Damgård, and V. Pastro. On the amortized complexity of zero knowledge protocols for multiplicative relations. In *ICITS*, 2012. To appear.
11. 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.
12. 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.
13. I. Damgård and C. Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In *CRYPTO*, pages 558–576, 2010.
14. N. Gama and P. Q. Nguyen. Predicting lattice reduction. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer, 2008.
15. C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.
16. 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.
17. 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.
18. 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.
19. Y. Lindell. Highly-efficient universally-composable commitments based on the ddh assumption. In *EUROCRYPT*, pages 446–466, 2011.
20. R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In A. Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011.
21. V. Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In M. Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616. Springer, 2009.
22. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. 2011. Manuscript.
23. D. Micciancio and O. Regev. Lattice-based cryptography, 2008.
24. S. Myers, M. Sergi, and abhi shelat. Threshold fully homomorphic encryption and secure computation. *IACR Cryptology ePrint Archive*, 2011:454, 2011.
25. 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.
26. M. Püschel and J. M. F. Moura. Algebraic signal processing theory: Cooley-tukey type algorithms for dcts and dsts. *IEEE Transactions on Signal Processing*, 56(4):1502–1521, 2008.

27. N. P. Smart and F. Vercauteren. Fully homomorphic simd operations. *IACR Cryptology ePrint Archive*, 2011:133, 2011.
28. S. Winkler and J. Wullschleger. On the efficiency of classical and quantum oblivious transfer reductions. In *CRYPTO*, pages 707–723, 2010.

A Proofs

A.1 Zero-Knowledge Proof

Construction of the Protocol. We will give two versions of the protocol. The first is a standard 3-move protocol, the second uses an "abort" technique to optimize the parameter values, this one is best suited for use with the Fiat-Shamir heuristic, and may be the best option for a practical implementation.

For the protocol, we will need that $\tau = p/2$, so that $\|\mathsf{encode}(\mathbf{m})\|_\infty \leq \tau = p/2$. This means that each entry in $\mathsf{encode}(\mathbf{m})$ corresponds to a uniquely determined residue mod p (or equivalently an element in $\mathbb{Z}_p$) and conversely each such residue is uniquely determined by $\mathbf{m}$. We did not ask for this in the abstract description, but the concrete instantiation satisfies this. Note that one problem we need to address in the protocol is that not all vectors in the input domain of decode will give us results in $\mathbb{F}_{p^k}$. However, if an input is equivalent mod p to $\mathsf{encode}(\mathbf{m})$ for some $\mathbf{m}$ then this is indeed the case, since then decode will return $\mathbf{m}$. Therefore the verifier explicitly checks whether the encodings the prover sends him decode to legal values, this will imply that the ciphertexts in question also decode to legal values.

We let R denote the matrix in $\mathbb{Z}^{\mathsf{sec}\times d}$ whose ith row is $\mathbf{r}_i$. It makes use of a matrix $M_{\mathbf{e}}$ defined as follows. Let $V := 2 \cdot \mathsf{sec} - 1$. For $\mathbf{e} \in \{0,1\}^{\mathsf{sec}}$ we define $M_{\mathbf{e}} \in \mathbb{Z}^{V\times \mathsf{sec}}$ to be the matrix whose (i,k)-th entry is given by $\mathbf{e}_{i-k+1}$, for $1 \leq i-k+1 \leq \mathsf{sec}$ and 0 otherwise.

Protocol Π_{ZKPoPK}

- For $i = 1,\ldots,V$, the prover sets $\mathbf{y}_i \leftarrow \mathbb{Z}^N$ and $\mathbf{s}_i \leftarrow \mathbb{Z}^d$, such that $\|\mathbf{y}_i\|_\infty \leq N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{\nu\mathsf{sec}-1}$ and $\|\mathbf{s}_i\|_\infty \leq d \cdot \rho \cdot \mathsf{sec}^2 \cdot 2^{\nu\mathsf{sec}-1}$. For $\mathbf{y}_i$, this is done as follows: choose a random message $\mathbf{m}_i \in (\mathbb{F}_{p^k})^s$ and set $\mathbf{y}_i = \mathsf{encode}(\mathbf{m}_i) + \mathbf{u}_i$, where each entry in $\mathbf{u}_i$ is a multiple of p, chosen uniformly at random, subject to $\|\mathbf{y}_i\|_\infty \leq N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{\nu\mathsf{sec}-1}$. If diag is set to true, then the $\mathbf{m}_i$ are chosen to be diagonal elements.
- The prover computes $a_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{y}_i, \mathbf{s}_i)$, for $i = 1,\ldots,V$, and defines $S \in \mathbb{Z}^{V\times d}$ to be the matrix whose ith row is $\mathbf{s}_i$ and sets $\mathbf{y} \leftarrow (\mathbf{y}_1,\ldots,\mathbf{y}_V)$, $\mathbf{a} \leftarrow (a_1,\ldots,a_V)$.
- The prover sends $\mathbf{a}$ to the verifier.
- The verifier selects $\mathbf{e} \in \{0,1\}^{\mathsf{sec}}$ and sends it to the prover.
- The prover sets $\mathbf{z} \leftarrow (\mathbf{z}_1,\ldots,\mathbf{z}_V)$, such that $\mathbf{z}^\mathsf{T} = \mathbf{y}^\mathsf{T} + M_{\mathbf{e}} \cdot \mathbf{x}^\mathsf{T}$, and $T = S + M_e \cdot R$. The prover sends $(\mathbf{z}, T)$ to the verifier.
- The verifier computes $d_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{z}_i, \mathbf{t}_i)$, for $i = 1,\ldots,V$, where $\mathbf{t}_i$ is the ith row of T and sets $\mathbf{d} \leftarrow (d_1,\ldots,d_V)$.
- The verifier checks that $\mathsf{decode}(\mathbf{z}_i) \in \mathbb{F}_{p^k}^s$ and whether the following three conditions hold; he rejects if not

$$\mathbf{d}^\mathsf{T} = \mathbf{a}^\mathsf{T} \boxplus \left(M_{\mathbf{e}} \boxtimes \mathbf{c}^\mathsf{T}\right), \quad \|\mathbf{z}_i\|_\infty \leq N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{\nu\mathsf{sec}-1}, \quad \|\mathbf{t}_i\|_\infty \leq d \cdot \rho \cdot \mathsf{sec}^2 \cdot 2^{\nu\mathsf{sec}-1}.$$

- If diag is set to true the verifier also checks whether $\mathsf{decode}(\mathbf{z}_i)$ is a diagonal element, and rejects if it is not.

Fig. 9. The ZKPoPK Protocol, interactive version.

Theorem 5. *The protocol* Π_{ZKPoPK} *(Appendix A.1, Figure 9) is an honest-verifier zero-knowledge proof of knowledge for the relation* R_{PoPK}*.*

Proof (Theorem 5).

Completeness: Assume the prover is honest. For $i = 1, \ldots, V$ the verifier checks if $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{z}_i, \mathbf{t}_i)$ equals $a_i \boxplus M_{\mathbf{e},i} \cdot \mathbf{c}^\mathsf{T}$, since $M_{\mathbf{e},i}$ is a scalar matrix we write multiplication with $\cdot$ as opposed to $\boxtimes$. The check passes because of the following relation:

$$\begin{aligned} a_i \boxplus \left(M_{\mathbf{e},i} \cdot \mathbf{c}^\mathsf{T}\right) &= \mathsf{Enc}_{\mathsf{pk}}(\mathbf{y}_i, \mathbf{s}_i) \boxplus_{k=1}^{\mathsf{sec}} (M_{\mathbf{e},i,k} \cdot c_k) \\ &= \mathsf{Enc}_{\mathsf{pk}}(\mathbf{y}_i, \mathbf{s}_i) \boxplus_{k=1}^{\mathsf{sec}} (M_{\mathbf{e},i,k} \cdot \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_k, \mathbf{r}_k)) \\ &= \mathsf{Enc}_{\mathsf{pk}}\left(\mathbf{y}_i + \sum_{k=1}^{\mathsf{sec}} M_{\mathbf{e},i,k} \cdot \mathbf{x}_k, \quad \mathbf{s}_i + \sum_{k=1}^{\mathsf{sec}} M_{\mathbf{e},i,k} \cdot \mathbf{r}_k\right) \\ &= \mathsf{Enc}_{\mathsf{pk}}\left(\mathbf{y}_i + M_{\mathbf{e},i} \cdot \mathbf{x}^\mathsf{T}, \quad \mathbf{s}_i + M_{\mathbf{e},i} \cdot \mathbf{r}^\mathsf{T}\right) = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{z}_i, \mathbf{t}_i). \end{aligned}$$

Moreover, given that $\mathbf{z}_i = \mathbf{y}_i + M_{\mathbf{e},i} \cdot \mathbf{x}^\mathsf{T}$ and that all ciphertexts in $\mathbf{c}$ are (τ, ρ)-ciphertexts, we get that each single coordinate in $M_{\mathbf{e},i} \cdot \mathbf{x}^\mathsf{T}$ is numerically at most $\mathsf{sec} \cdot \tau$. Each coordinate of $\mathbf{y}_i$ was chosen from an interval that is a factor $N \cdot \mathsf{sec} \cdot 2^{\nu\mathsf{sec}-1}$ larger. By a union bound bound over the $N \cdot \mathsf{sec}$ coordinates involved, each coordinate in $\mathbf{z}_i$ fails to be in the required range with probability exponentially small in sec. A similar argument shows that the check $\|\mathbf{t}_i\|_\infty$ also fails with negligible probability. Finally, each $\mathbf{y}_i$ was constructed to be congruent mod p to the encoding of a value in $\mathbb{F}_{p^k}^s$. Since this is also the case for the $\mathbf{x}_i$'s if the prover is honest, the same is true for the $\mathbf{z}_i$'s, and they therefore decode to a value in $\mathbb{F}_{p^k}^s$. If diag was set to true, all $\mathbf{x}_i, \mathbf{y}_i$ contain diagonal plaintexts, and then the same is true for the $\mathbf{z}_i$.

Soundness: We consider a prover making a verifier accept both $(x, \mathbf{a}, \mathbf{e}, (\mathbf{z}, T))$ and $(x, \mathbf{a}, \mathbf{e}', (\mathbf{z}', T'))$ with $\mathbf{e} \neq \mathbf{e}'$. Since both checks $\mathbf{d}^\mathsf{T} = \mathbf{a}^\mathsf{T} \boxplus (M_\mathbf{e} \cdot \mathbf{c}^\mathsf{T})$ and $\mathbf{d}'^\mathsf{T} = \mathbf{a}^\mathsf{T} \boxplus (M_{\mathbf{e}'} \cdot \mathbf{c}^\mathsf{T})$ passed, one can subtract the two equalities and obtain

$$(M_\mathbf{e} - M_{\mathbf{e}'}) \boxtimes \mathbf{c}^\mathsf{T} = \left(\mathbf{d} \boxminus \mathbf{d}'\right)^\mathsf{T} \tag{1}$$

In order to find $\mathbf{x}$ and R such that $c_k = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_k, \mathbf{r}_k)$ for $k = 1, \ldots, \mathsf{sec}$, we first solve (1) as a linear system in $\mathbf{c}$. Let j be the highest index such that $\mathbf{e}_j \neq \mathbf{e}'_j$. The $\mathsf{sec} \times \mathsf{sec}$ submatrix of $M_\mathbf{e} - M_{\mathbf{e}'}$, consisting of the rows of $M_\mathbf{e} - M_{\mathbf{e}'}$ between j and $j + \mathsf{sec} - 1$ both included, is upper triangular with entries in $\{-1, 0, 1\}$ and its diagonal consists of the non-zero value $\mathbf{e}_j - \mathbf{e}'_j$ (so it is possible to find a solution for $\mathbf{c}$). Since the verifier has values $\mathbf{z}_i, \mathbf{t}_i, \mathbf{z}'_i, \mathbf{t}'_i$ such that $d_i = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{z}_i, \mathbf{t}_i)$ and $d'_i = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{z}'_i, \mathbf{t}'_i)$, and given that $c_i = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_i, \mathbf{r}_i)$, it is possible to directly solve the linear system in $\mathbf{x}$ and R (since the cryptosystem is additively homomorphic), from the bottom equation to the one "in the middle" with index $\mathsf{sec}/2$. Since $\|\mathbf{z}_i\|_\infty, \|\mathbf{z}'_i\|_\infty \leq N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{\nu\mathsf{sec}-1}$ and $\|\mathbf{t}_i\|_\infty, \|\mathbf{t}'_i\|_\infty \leq d \cdot \rho \cdot \mathsf{sec}^2 \cdot 2^{\nu\mathsf{sec}-1}$, we conclude that $c_{\mathsf{sec}-i}$ is a $(s \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{\nu\mathsf{sec}+i}, d \cdot \rho \cdot \mathsf{sec}^2 \cdot 2^{\nu\mathsf{sec}+i})$-ciphertext (by induction on i). To solve for $c_1, \ldots c_{\mathsf{sec}/2}$, we consider the *lowest* index j such that $\mathbf{e}_j \neq \mathbf{e}'_j$, construct an lower triangular matrix in a similar way as above, and solve from the first equation downwards. We conclude that $\mathbf{c}$ contains $(N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{(1/2+\nu)\mathsf{sec}}, d \cdot \rho \cdot \mathsf{sec}^2 \cdot 2^{(1/2+\nu)\mathsf{sec}})$-ciphertexts.

We note that since the verifier accepted, each $\mathbf{z}_i$ has small norm and decodes to a value in $(\mathbb{F}_{p^k})^s$. Since we can write $\mathbf{x}_i$ as a linear combination of the $\mathbf{z}_i$, it follows from correctness of the cryptosystem that the $\mathbf{x}_i$ also decode to values in $(\mathbb{F}_{p^k})^s$. Finally, if diag was set to true, the verifier only accepts if all $\mathbf{z}_i$ decode to diagonal values. Again, since we can write $\mathbf{x}_i$ as a linear combination of the $\mathbf{z}_i$, the $\mathbf{x}_i$ also decode to diagonal values.

Zero-Knowledge: We give an honest-verifier simulator for the protocol that outputs accepting conversations. In order to simulate one repetition, the simulator samples $\mathbf{e} \in \{0,1\}^{\mathsf{sec}}$ uniformly and $\mathbf{z}, T$ uniformly with the constrain that $\mathbf{d}$ contains random ciphertexts satisfying the verifiers check, i.e., $\mathbf{z}_i, \mathbf{t}_i$ are uniform, subject to $\|\mathbf{z}_i\|_\infty \leq N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{\nu\mathsf{sec}-1}$, $\|\mathbf{t}_i\|_\infty \leq d \cdot \rho \cdot \mathsf{sec}^2 \cdot 2^{\nu\mathsf{sec}-1}$, where moreover $\mathbf{z}_i$ is generated as $\mathsf{encode}(\mathbf{m}_i)+\mathbf{u}_i$ where $\mathbf{m}_i$ is a random plaintext (diagonal if diag is set to true) and $\mathbf{u}_i$ contains multiples of p that are uniformly random, subject to $\|\mathbf{z}_i\|_\infty \leq N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{\nu\mathsf{sec}-1}$. Finally, $\mathbf{a}$ is computed as $\mathbf{a}^\mathsf{T} \leftarrow \mathbf{d}^\mathsf{T} \boxminus (M_\mathbf{e} \cdot \mathbf{c}^\mathsf{T})$. In the real conversation, the provers choice of values in $\mathbf{z}_i$ and $\mathbf{t}_i$ are statistically close to the distribution used by the simulator. This is because the prover uses the same method to generate these values, except that he adds in some vectors of exponentially smaller norm which leads to a statistically close distribution. Since $\mathbf{e}$ has the correct distribution and $\mathbf{a}$ follows deterministically from the last two messages, the simulation is statistically indistinguishable.

□

We now give a protocol that leads to smaller values of the parameters and hence also allows better parameters for the underlying cryptoystem. This version, however, is better suited for use with the Fiat-Shamir heuristic. The idea is to let the prover choose his randomness in a smaller interval, and abort if the last message would reveal too much information. This is an idea from [21]. When using the Fiat-Shamir heuristic, this is not a problem as he prover only needs to show a successful attempt to he verifier. We let h be a suitable hash function that outputs sec-bit strings.

Protocol Π_{ZKPoPK}

- For $i = 1, \ldots, V$, the prover generates $\mathbf{y}_i \leftarrow \mathbb{Z}^N$ and $\mathbf{s}_i \leftarrow \mathbb{Z}^d$, such that $\|\mathbf{y}_i\|_\infty \leq 128 \cdot N \cdot \tau \cdot \mathsf{sec}^2$ and $\|\mathbf{s}_i\|_\infty \leq 128 \cdot d \cdot \rho \cdot \mathsf{sec}^2$. For $\mathbf{y}_i$, this is done as follows: choose a random message $\mathbf{m}_i \in (\mathbb{F}_{p^k})^s$ and set $\mathbf{y}_i = \mathsf{encode}(\mathbf{m}_i) + \mathbf{u}_i$, where each entry in $\mathbf{u}_i$ is a multiple of p, chosen uniformly at random, subject to $\|\mathbf{y}_i\|_\infty \leq 128 \cdot N \cdot \tau \cdot \mathsf{sec}^2$. If diag is set to true then the $\mathbf{m}_i$ are additionally chosen to be diagonal elements.
- The prover computes $a_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{y}_i, \mathbf{s}_i)$, for $i = 1, \ldots, V$, and defines $S \in \mathbb{Z}^{V \times d}$ to be the matrix whose ith row is $\mathbf{s}_i$ and sets $\mathbf{y} \leftarrow (\mathbf{y}_1, \ldots, \mathbf{y}_V)$, $\mathbf{a} \leftarrow (a_1, \ldots, a_V)$.
- The prover sends $\mathbf{a}$ to the verifier.
- The prover computes $\mathbf{e} = h(\mathbf{a}, \mathbf{c})$.
- The prover sets $\mathbf{z} \leftarrow (\mathbf{z}_1, \ldots, \mathbf{z}_V)$, such that $\mathbf{z}^\mathsf{T} = \mathbf{y}^\mathsf{T} + M_\mathbf{e} \cdot \mathbf{x}^\mathsf{T}$, and $T = S + M_e \cdot R$. Let $\mathbf{t}_i$ be the ith row of T. If for any i, it is the case that $\|\mathbf{z}_i\|_\infty > 128 \cdot N \cdot \tau \cdot \mathsf{sec}^2 - \tau \cdot \mathsf{sec}$ or $\|\mathbf{t}_i\|_\infty > 128 \cdot d \cdot \rho \cdot \mathsf{sec}^2 - \rho \cdot \mathsf{sec}$, the prover aborts and the protocol is restarted. Otherwise the prover sends $(\mathbf{a}, \mathbf{z}, T)$ to the verifier.
- The verifier computes $\mathbf{e} = h(\mathbf{a}, \mathbf{c})$, $d_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{z}_i, \mathbf{t}_i)$, for $i = 1, \ldots, V$, where $\mathbf{t}_i$ is the ith row of T and sets $\mathbf{d} \leftarrow (d_1, \ldots, d_V)$.
- The verifier checks $\mathsf{decode}(\mathbf{z}_i) \in \mathbb{F}_{p^k}^s$ and whether the following three conditions hold

$$\mathbf{d}^\mathsf{T} = \mathbf{a}^\mathsf{T} \boxplus \left(M_\mathbf{e} \boxtimes \mathbf{c}^\mathsf{T}\right), \quad \|\mathbf{z}_i\|_\infty \leq 128 \cdot N \cdot \tau \cdot \mathsf{sec}^2, \quad \|\mathbf{t}_i\|_\infty \leq 128 \cdot d \cdot \rho \cdot \mathsf{sec}^2.$$

If diag is set to true the verifier also checks whether $\mathsf{decode}(\mathbf{z}_i)$ is a diagonal element, and rejects if it is not.

Fig. 10. The ZKPoPK Protocol, version for Fiat-Shamir heuristic.

We claim that the Fiat-Shamir based protocol is a proof of knowledge for the relation in question in the random oracle model. In this case, however, we can guarantee that the adversarially generated ciphertexts are $(N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{\mathsf{sec}/2+8}, d \cdot \rho \cdot \mathsf{sec}^2 \cdot 2^{\mathsf{sec}/2+8})$- ciphertexts.

Completeness: Assume the prover is honest. Note first that each $\mathbf{y}_i$ was constructed to be congruent mod p to the encoding of a value in $(\mathbb{F}_{p^k})^s$. Since this is also the case for the $\mathbf{x}_i$'s if the prover is

honest, the same is true for the $\mathbf{z}_i$'s, and they therefore always decode to a value in $(\mathbb{F}_{p^k})^s$. If diag was set to true, all $\mathbf{x}_i, \mathbf{y}_i$ contain diagonal plaintexts, and then the same is true for the $\mathbf{z}_i$.

Next, for $i = 1, \ldots, V$ the verifier checks if $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{z}_i, \mathbf{t}_i)$ equals $a_i \boxplus M_{\mathbf{e},i} \cdot \mathbf{c}^\mathsf{T}$, since $M_{\mathbf{e},i}$ is a scalar matrix we write multiplication with $\cdot$ as opposed to $\boxtimes$. The check passes because of the following relation:

$$\begin{aligned}
a_i \boxplus \left(M_{\mathbf{e},i} \cdot \mathbf{c}^\mathsf{T}\right) &= \mathsf{Enc}_{\mathsf{pk}}(\mathbf{y}_i, \mathbf{s}_i) \boxplus_{k=1}^{\mathsf{sec}} (M_{\mathbf{e},i,k} \cdot c_k) \\
&= \mathsf{Enc}_{\mathsf{pk}}(\mathbf{y}_i, \mathbf{s}_i) \boxplus_{k=1}^{\mathsf{sec}} (M_{\mathbf{e},i,k} \cdot \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_k, \mathbf{r}_k)) \\
&= \mathsf{Enc}_{\mathsf{pk}}\left(\mathbf{y}_i + \sum_{k=1}^{\mathsf{sec}} M_{\mathbf{e},i,k} \cdot \mathbf{x}_k, \quad \mathbf{s}_i + \sum_{k=1}^{\mathsf{sec}} M_{\mathbf{e},i,k} \cdot \mathbf{r}_k\right) \\
&= \mathsf{Enc}_{\mathsf{pk}}\left(\mathbf{y}_i + M_{\mathbf{e},i} \cdot \mathbf{x}^\mathsf{T}, \quad \mathbf{s}_i + M_{\mathbf{e},i} \cdot \mathbf{r}^\mathsf{T}\right) = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{z}_i, \mathbf{t}_i).
\end{aligned}$$

Moreover, given that $\mathbf{z}_i = \mathbf{y}_i + M_{\mathbf{e},i} \cdot \mathbf{x}^\mathsf{T}$ and that all ciphertexts in $\mathbf{c}$ are (τ, ρ)-ciphertexts, we get that each single coordinate in $M_{\mathbf{e},i} \cdot \mathbf{x}^\mathsf{T}$ is numerically at most $\mathsf{sec} \cdot \tau$. Each coordinate of $\mathbf{y}_i$ was chosen from an interval that is a factor $128 \cdot N \cdot \mathsf{sec}$ larger. Therefore each coordinate in $\mathbf{z}_i$ fails to be in the required range with probability $1/(128 \cdot N \cdot \mathsf{sec})$. Note that this probability does not depend on the concrete values of the coordinates in $M_{\mathbf{e},i} \cdot \mathbf{x}^\mathsf{T}$, only on the bound on the numeric value.

By a union bound over the N coordinates of $\mathbf{z}_i$ we get that $\|\mathbf{z}_i\|_\infty \leq 128 \cdot N \cdot \tau \cdot \mathsf{sec}^2 - \tau \cdot \mathsf{sec}$ fails with probability at most $1/(128 \cdot \mathsf{sec})$, and by a final union bound over the $2\,\mathsf{sec} - 1$ ciphtertexts that all checks on the $\mathbf{z}_i$'s are ok except with probability at most $1/64$. A similar argument shows that the check $\|\mathbf{t}_i\|_\infty \leq 128 \cdot d \cdot \rho \cdot \mathsf{sec}^2 - \rho \cdot \mathsf{sec}$ fails also with probability at most $1/64$. The conclusion is that the prover will abort with probability at most $1/32$, so we expect to only have to repeat the protocol once to have success.

Soundness: By a standard argument, a prover who can efficiently produce a valid proof is able to produce $(x, \mathbf{a}, \mathbf{e}, (\mathbf{z}, T))$ and $(x, \mathbf{a}, \mathbf{e}', (\mathbf{z}', T'))$ with $\mathbf{e} \neq \mathbf{e}'$ that the verifier would accept. Since both checks $\mathbf{d}^\mathsf{T} = \mathbf{a}^\mathsf{T} \boxplus (M_{\mathbf{e}} \cdot \mathbf{c}^\mathsf{T})$ and $\mathbf{d}'^\mathsf{T} = \mathbf{a}^\mathsf{T} \boxplus (M_{\mathbf{e}'} \cdot \mathbf{c}^\mathsf{T})$ passed, one can subtract the two equalities and obtain

$$(M_{\mathbf{e}} - M_{\mathbf{e}'}) \boxtimes \mathbf{c}^\mathsf{T} = \left(\mathbf{d} \boxminus \mathbf{d}'\right)^\mathsf{T} \tag{2}$$

In order to find $\mathbf{x}$ and R such that $c_k = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_k, \mathbf{r}_k)$ for $k = 1, \ldots, \mathsf{sec}$, we first solve (2) as a linear system in $\mathbf{c}$. Let j be the highest index such that $\mathbf{e}_j \neq \mathbf{e}'_j$. The $\mathsf{sec} \times \mathsf{sec}$ submatrix of $M_{\mathbf{e}} - M_{\mathbf{e}'}$, consisting of the rows of $M_{\mathbf{e}} - M_{\mathbf{e}'}$ between j and $j + \mathsf{sec} - 1$ both included, is upper triangular with entries in $\{-1, 0, 1\}$ and its diagonal consists of the non-zero value $\mathbf{e}_j - \mathbf{e}'_j$ (so it is possible to find a solution for $\mathbf{c}$). Since the verifier has values $\mathbf{z}_i, \mathbf{t}_i, \mathbf{z}'_i, \mathbf{t}'_i$ such that $d_i = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{z}_i, \mathbf{t}_i)$ and $d'_i = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{z}'_i, \mathbf{t}'_i)$, and given that $c_i = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_i, \mathbf{r}_i)$, it is possible to directly solve the linear system in $\mathbf{x}$ and R (since the cryptosystem is additively homomorphic), from the bottom equation to the one "in the middle" with index $\mathsf{sec}/2$.
Since $\|\mathbf{z}_i\|_\infty, \|\mathbf{z}'_i\|_\infty \leq 128 \cdot N \cdot \tau \cdot \mathsf{sec}^2$ and $\|\mathbf{t}_i\|_\infty, \|\mathbf{t}'_i\|_\infty \leq 128 \cdot d \cdot \rho \cdot \mathsf{sec}^2$, we conclude that $c_{\mathsf{sec}-i}$ must be a $(256 \cdot N \cdot \tau \cdot 2^i \cdot \mathsf{sec}^2, 256 \cdot d \cdot \rho \cdot 2^i \cdot \mathsf{sec}^2)$-ciphertext (by induction on i). To solve for $c_1, \ldots c_{\mathsf{sec}/2}$, we consider the *lowest* index j such that $\mathbf{e}_j \neq \mathbf{e}'_j$, construct an lower triangular matrix in a similar way as above, and solve from the first equation downwards. We conclude that $\mathbf{c}$ contains $(N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{sec/2+8}, d \cdot \rho \cdot \mathsf{sec}^2 \cdot 2^{sec/2+8})$-ciphertexts.

We note that since the verifier accepted, each $\mathbf{z}_i$ has small norm and decodes to a value in $(\mathbb{F}_{p^k})^s$. Since we can write $\mathbf{x}_i$ as a linear combination of the $\mathbf{z}_i$, it follows from correctness of the cryptosystem that the $\mathbf{x}_i$ also decode to values $(\mathbb{F}_{p^k})^s$. Finally, if diag was set to true, the verifier only accepts if all $\mathbf{z}_i$ decode to diagonal values. Again, since we can write $\mathbf{x}_i$ as a linear combination of the $\mathbf{z}_i$, the $\mathbf{x}_i$ also decode to diagonal values.

Zero-Knowledge: We give an honest-verifier simulator for the protocol that outputs an accepting conversation (that does not abort).

In order to simulate one repetition, the simulator samples $\mathbf{e} \in \{0,1\}^{\mathsf{sec}}$ uniformly and $\mathbf{z}, T$ uniformly with the constrain that $\mathbf{d}$ contains random $(8 \cdot N \cdot \tau \cdot \mathsf{sec}^2 - \tau \cdot \mathsf{sec}, 8 \cdot d \cdot \rho \cdot \mathsf{sec}^2 - \rho \cdot \mathsf{sec})$-ciphertexts. where moreover $\mathbf{z}_i$ is generated as $\mathsf{encode}(\mathbf{m}_i) + \mathbf{u}_i$ where $\mathbf{m}_i$ is a random plaintext (a diagonal one if diag is set to true) and $\mathbf{u}_i$ contains multiples of p that are uniformly random, subject to $\|\mathbf{z}_i\|_\infty \leq 8N \cdot \tau \cdot \mathsf{sec}^2 - \tau \cdot \mathsf{sec}$. Finally, $\mathbf{a}$ is computed as $\mathbf{a}^\mathsf{T} \leftarrow \mathbf{d}^\mathsf{T} \boxminus (M_\mathbf{e} \cdot \mathbf{c}^\mathsf{T})$. Define the random oracle to output $\mathbf{e}$ on input $\mathbf{a}, \mathbf{c}$, output $(\mathbf{a}, \mathbf{e}, (\mathbf{z}, T))$ and stop.

We argue that this simulation is perfect: The distribution of a simulated $\mathbf{e}$ is the same as a real one. Also, it is straightforward to see that in a real conversation, given that the prover does not abort, the vectors $\mathbf{z}_i, \mathbf{t}_i$ will be uniformly random, subject to $\|\mathbf{z}_i\|_\infty \leq 8 \cdot s \cdot \tau \cdot \mathsf{sec}^2 - \tau \cdot \mathsf{sec}$ and $\|\mathbf{t}_i\|_\infty \leq 8 \cdot d \cdot \rho \cdot \mathsf{sec}^2 - \rho \cdot \mathsf{sec}$. So the simulator chooses $\mathbf{z}_i, \mathbf{t}_i$ with exactly the right distribution. Since the value of $\mathbf{a}$ follows deterministically from the $\mathbf{e}, \mathbf{z}_i, \mathbf{t}_i$, we have what we wanted.

Doing without random oracles. The above protocol can also be executed without using the Fiat-Shamir heuristic. In this case, the prover will start $\mathsf{sec}/5$ instances of the protocol, computing $\mathbf{a}_1, \ldots, \mathbf{a}_{\mathsf{sec}/5}$. We choose this number of instance because it will ensure that the prover fails on all of them with probability only $(1/32)^{\mathsf{sec}/5} = 2^{-\mathsf{sec}}$. The prover commits to all these values, which can be done, for instance, with a Merkle hash tree, in which case the commitment will be very short, and any of $\mathbf{a}$'s can be opened by sending a piece of information that is only logarithmic in sec.

The verifier selects $\mathbf{e}$, the prover finds an instance where he would not abort the protocol with this $\mathbf{e}$, opens the corresponding $\mathbf{a}$ and completes that instance.

This is complete and zero-knowledge by the same argument as above plus the hiding property of the commitment scheme used. Soundness follows from the fact that if the prover succeeds with probability significantly greater that $2^{-\mathsf{sec}} \cdot \mathsf{sec}/5$ he must be able to answer different challenges correctly for some fixed instance out of the $\mathsf{sec}/5$ we have. Such answers can be extracted by rewinding, and then the rest of the argument is the same as above.

A.2 The UC Model

In the following sections, we show that the online and preprocessing phases of our protocol are secure in the UC model. We briefly recall how this model works: we will use the variant where there is only one adversarial entity, the environment $\mathcal{Z}$. The environment chooses inputs for the honest players and gets their outputs when the protocol is done. It also does an attack on the protocol which is our case means that it corrupts up to $n-1$ of the players and takes control over their actions. When $\mathcal{Z}$ stops, it outputs a bit. This process where $\mathcal{Z}$ interacts with the real players and protocol is called the real process.

To define what it means that the protocol implements functionality $\mathcal{F}$ securely we assume there exists a simulator $\mathcal{S}$ that interacts with both $\mathcal{F}$ and $\mathcal{Z}$. Towards $\mathcal{F}$, it chooses inputs for the corrupt

players and will get their outputs. Towards $\mathcal{Z}$, it must simulate a view of the protocol that looks like what $\mathcal{Z}$ would see in a real attack. This process is called the ideal process, and here $\mathcal{F}$ supplies $\mathcal{Z}$ with the i/o interface of honest players. We say that the protocol implements $\mathcal{F}$ securely if $\mathcal{Z}$ outputs 1 with essentially the same probability in the real as in the ideal process. We speak of computational security if $\mathcal{Z}$ is assumed to be poly-time bounded and of statistical security if $\mathcal{Z}$ is unbounded.

A.3 Online Phase

On generating the e_i's Before proving the online protocol UC secure, we compute the probability of getting away with cheating in step 4 of 'Output' and how this depends on the way we generate the e_i's.

For this purpose we design the following security game:

1. The challenger generates the secret key α and MACs $\gamma_i \leftarrow \alpha m_i$ and sends messages $m_1, \ldots, m_T$ to the adversary.
2. The adversary sends back messages $m'_1, \ldots, m'_T$.
3. The challenger generates random values $e_1, \ldots, e_T \leftarrow \mathbb{F}_{p^k}$ and sends them to the adversary.
4. The adversary provides an error Δ.
5. Set $m \leftarrow \sum_{i=0}^{T} e_i m'_i, \gamma \leftarrow \sum_{i=0}^{T} e_i \gamma_i$. Now, the challenger checks that $\alpha m = \gamma + \Delta$

The adversary wins the game if there is an i for which $m'_i \neq m_i$ and the final check goes through.

It is not difficult to see that this game indeed models 'Output'(up to step 4): The second step in the game where the adversary sends the m'_i's models the fact that corrupted players can choose to lie about their shares of values opened during the protocol execution. Δ models the fact that the adversary is allowed to introduce errors on the macs when data are sent to $\mathcal{F}_{\text{PREP}}$ in the initial part of the protocol and may also modify the shares of macs held by corrupt players. Finally, since α, γ are secret shared in the protocol, the adversary has no information on α, γ ahead of time in the protocol, just as in the security game.

Now, let us look at the probability of winning the game if the e_i's are randomly chosen. If the check goes through, we have that the following equality holds: $\alpha \sum_{i=0}^{T} e_i(m'_i - m_i) = \Delta$. First we consider the case where $\sum_{i=0}^{T} e_i(m'_i - m_i) \neq 0$, so $\alpha = \Delta / \sum_{i=0}^{T} e_i(m'_i - m_i)$. This implies that being able to pass the check is equivalent to guessing α. However, since the adversary has no information about α, this happens with probability only $1/|\mathbb{F}_{p^k}|$. So what is left is to argue that $\sum_{i=0}^{T} e_i(m'_i - m_i) = 0$ also happens with very low probability. This can be seen as follows. We define $\mu_i := (m'_i - m_i)$ and $\mu := (\mu_1, \ldots, \mu_T), e := (e_1, \ldots, e_T)$. Now $f_\mu(e) := e \cdot \mu = \sum_{i=0}^{T} e_i \mu_i$ defines a linear mapping, which is not the 0-mapping since at least one $\mu_i \neq 0$. From linear algebra we then have the rank-nullity theorem telling us that $\dim(\ker(f_\mu)) = T - 1$. Also since e is random and the adversary does not know e when choosing the m'_i's, the probability of $e \in \ker(f_\mu)$ is $|\mathbb{F}_{p^k}^{T-1}|/|\mathbb{F}_{p^k}^{T}| = 1/|\mathbb{F}_{p^k}|$. Summing up, the total probability of winning the game is at most $2/|\mathbb{F}_{p^k}|$.

Since choosing the e_i's uniformly would require an expensive coin-flip protocol, we use a different way to generate them in the protocol: namely e_1 is chosen at random and for $i > 1$, $e_i \leftarrow e_1^i$. This has the advantage of adding only a constant number of multiplications in $\mathbb{F}_{p^k}$ for a secure multiplication. On the security side, we still want that $\sum_{i=0}^{T} e_i \mu_i = 0$ should happen with small probability. Viewing f_μ as a polynomial of degree T, we know it has at most T roots, so we have to make sure we have an upper bound on T such that e_1 is chosen from a field big enough for T/p^k to be negligible.

An alternative approach would be to use a pseudorandom generator G. We would then have shared some random seed $\langle s \rangle$. By opening $\langle s \rangle$ and feeding it to G we can generate T pseudorandom elements. In the protocol, the parties would commit to their share of the MAC on s, and when α becomes public, the MAC would be checked. If it is OK, the protocol would go on with the rest of the checks. With respect to cheating the argument is basically the same; If an adversary A has a significant probability of choosing m_i''s such that $\sum_{i=0}^{T} e_i(m_i' - m_i) = 0$, then the G is a bad pseudorandom generator, or in other words, we can use A to break G. With this way of generating the e_i's, we increase the complexity for one secure multiplication by whatever G needs to generate one pseudorandom element.

Proof (Theorem 1). We construct a simulator $\mathcal{S}_{\text{AMPC}}$ such that a poly-time environment $\mathcal{Z}$ cannot distinguish between the real protocol system and the ideal. We assume here static, active corruption. The simulator runs a copy of the protocol Π_{ONLINE} and simulates the ideal functionalities for preprocessing and commitment. It relays messages between parties/$\mathcal{F}_{\text{PREP}}$ and $\mathcal{Z}$, such that $\mathcal{Z}$ will see the same interface as when interacting with a real protocol. The specification of the simulator $\mathcal{S}_{\text{AMPC}}$ is presented in Figure 11.

Simulator $\mathcal{S}_{\text{AMPC}}$

Initialize: The simulator creates the desired number of triples by doing the steps in $\mathcal{F}_{\text{PREP}}$. Note that here the simulator will read all data of the corrupted parties specified to the copy of $\mathcal{F}_{\text{PREP}}$.

Rand: The simulator runs the copy protocol honestly and calls *rand* on the ideal functionality $\mathcal{F}_{\text{AMPC}}$.

Input: If P_i is not corrupted the copy is run honestly with dummy input, for example 0.
If P_i is corrupted the input step is done honestly and then the simulator waits for P_i to broadcast δ. Given this, the simulator can compute $x_i' \leftarrow (r + \delta)$ since it knows (all the shares of) r. This is the supposed input of P_i, which the simulator now gives to the ideal functionality $\mathcal{F}_{\text{AMPC}}$.

Add: The simulator runs the protocol honestly and calls *add* on the ideal functionality $\mathcal{F}_{\text{AMPC}}$.

Multiply: The simulator runs the protocol honestly and calls *multiply* on the ideal functionality $\mathcal{F}_{\text{AMPC}}$.

Output: The output step is run and the protocol is aborted if one of the checks in step 4 does not go through. Otherwise the simulator calls *output* on $\mathcal{F}_{\text{AMPC}}$ and gets the result y back. Now it has to simulate shares y_j of honest parties such that they are consistent with y. Note that the simulator already has shares of an output value y' that was computed using the dummy inputs, as well as shares of the MAC for y'. The simulator now selects an honest party, say P_k and adds $y - y'$ to his share of y and $\alpha(y - y')$ to his share of the MAC. Note that the simulator can compute $\alpha(y - y')$ since it knows from the beginning (all the shares of) α. Now it simulates the openings of shares of y towards the environment according to the protocol. If this terminates correctly, send OK to $\mathcal{F}_{\text{AMPC}}$ (causing it to output y to the honest players).

Fig. 11. The simulator for $\mathcal{F}_{\text{AMPC}}$.

To see that the simulated and real processes cannot be distinguished, we will show that the view of the environment in the ideal process is statistically indistinguishable from the view in the real process. This view consists of the corrupt players' view of the protocol execution as well as the inputs and outputs of honest players.

We first argue that the view up to the point where the output value is opened (step 5 of the 'output' stage of the protocol) has exactly the same distribution in the real and in the simulated case: First, the value broadcast by honest players in the input stage are always uniformly random. Second, when a value is partially opened in a secure multiplication, fresh shares of a random value are subtracted, so the honest players will always send a set of uniformly random and independent values. Third, the honest players hold shares in MACs on the opened values, these are random sharings of a correct MAC with an error added that is determined by the errors specified by the

environment in the initial phase. Therefore, also the MAC and shares revealed in step 4 of 'output' have the same distribution in the simulated as in the the real process. Finally note that if the simulated protocol aborts, the simulator makes the ideal functionality fail, so the environment will see that honest players generate no output, just as when the real process aborts.

Now, if the real or simulated protocol proceeds to the last step, the only new data that the environment sees is an output value y, plus some shares of honest players. These are random shares that are consistent with y and its MAC in both the simulated and real case. In other words, the environments' view of the last step has the same distribution in real and simulated case as long as y is the same.

In the simulation, y is of course the correct evaluation on the inputs matching the shares that were read from the corrupted parties in the beginning. To finish the proof, it is therefore sufficient to show that the same happens in the real process with overwhelming probability. In other words, the event that the real protocol terminates but the output is not correct occurs with negligible probability.

Incorrect outputs can result either from corrupted parties who during the protocol successfully cheat with their shares or from having computed with triples where the multiplicative relation does not hold (even if the revealed shares were correct). For the latter case we argue that with correct shares the multiplicative relation holds with overwhelming probability, and this follows from the check on the triples in step 1 of 'Multiply': It is easy to see that if the triples are correct, the check will be true. On the other hand, if some triple is not correct, (in spite of correct shares), the probability of satisfying the check is $1/|\mathbb{F}_{p^k}|$, since there is only one random challenge t, for which $t \cdot (c - a \cdot b) = (h - g \cdot f)$. For the former case regarding the checking of shares, we have checks related to the openings of $[\![\cdot]\!]$-values (during 'Input and a single one in 'Output'). The rest of the checking is done in steps 4 and 5 of 'Output'. Being able to cheat during an opening of a $[\![\ cdot]\!]$-value corresponds to guessing at least one private key β_i. Assuming β_i is chosen randomly in $\mathbb{F}_{p^k}$, the probability is at most $1/|\mathbb{F}_{p^k}|$. Furthermore, as we discussed in the beginning of this section, the probability of a party being able to cheat in step 4 is $(T+1)/|\mathbb{F}_{p^k}|$ where T is the number of values opened during secure multiplications. In step 5, only one MAC is checked for each output, so here the probability of cheating is $1/|\mathbb{F}_{p^k}|$ per check as argued earlier. Since the protocol aborts as soon as a check fails, the probability that it terminates with an incorrect output is the maximum probability with which any single check can be cheated, which in our case is $(T+1)/|\mathbb{F}_{p^k}|$. This is negligible, since we assume that T is polynomial while p^k is exponential in sec. □

Commitments based on $\mathcal{F}_{\text{PREP}}$ In the above we assumed access to an ideal functionality for commitments. We can, however, do the commitments needed in our protocol based only on the output of $\mathcal{F}_{\text{PREP}}$ as follows. First a random value $[\![r]\!]$ is opened to the committer P_i (This could even be done in the preprocessing). To commit to a value x, P_i broadcasts $c = r + x$. To open the commitment, $[\![r]\!]$ is opened to all the players who can now compute $c - r = x$. Correctness is still guaranteed because of the MACs in $[\![r]\!]$. Furthermore, since to begin with $[\![r]\!]$ is only opened to P_i, we have that c is indistinguishable from a random value and can thus easily be simulated. To simulate during 'Output' when P_i is honest and has to open his commitment, the simulator simply changes P_i's share of $[\![r]\!]$ and the shares of the MACs to make it fit with the broadcasted value and the value he should have committed to. This is possible because the simulator knows all MAC keys. It is easy to see that this has communication and computational complexity $O(n^2)$ per commitment.

Implementing Broadcast and Multiple Inputs/Outputs To implement broadcast based on point-to-point channels, we first observe that since we do not guarantee termination anyway, the broadcast does not have to terminate either. Therefore the following very simple protocol for broadcasting $x \in \mathbb{F}_{p^k}$ is sufficient:

1. The broadcaster sends x to all players.
2. Each player sends to all players what he received in the previous step.
3. Each player checks that he received the value x from all players. If, so output x, otherwise abort.

This protocol has communication complexity $O(n^2)$ field elements for one broadcast. However, this can be optimized in case we need to broadcast many values. Below, we assume each player sends one value, say P_i wants to send x_i. We also assume that we have a random value $[\![s]\!]$ from the preprocessing, and that we have an ϵ-almost universal class of hash functions $\{h_s\}$ for negligible ϵ, indexed by values s, taking as input strings of n elements in $\mathbb{F}_{p^k}$ and producing output in $\mathbb{F}_{p^k}$. A simple example is where we view the input F as specifying coefficients of a polynomial of degree $n-1$, and $h_s(F)$ is the result of evaluating this polynomial in point s. If two inputs F, F' are distinct, their difference has at most $n-1$ roots, so the probability that $h_s(F) = h_s(F')$ is $(n-1)/p^k$. The protocol goes as follows:

1. P_i sends x_i to all players.
2. $[\![s]\!]$ is opened.
3. Each player sends to all players $h_s(F)$ where F is the string of values he received in the first step.
4. Each players checks that he received the same hash value from all players. If, so output $x_1, \ldots, x_n$ as received in the first step, otherwise abort.

It is clear that if a player sent different data to different honest players, some honest player will abort, except with probability $(n-1)/p^k$. This protocol has complexity $O(n^2)$, including also the cost of opening $[\![s]\!]$. But the cost per value we broadcast is only $O(n)$. This protocol generalizes easily to a case where one player has n values to broadcast.

In the online protocol we specified before, broadcast is used to give inputs in the first stage. Here, all players broadcast a value, and this is readily implemented with the optimized broadcast protocol above, so we get complexity $O(n)$ per input gate. If players have several inputs, we just execute several instances of this broadcast.

The only other point where broadcast is used is in partial openings where a designated player P_1 broadcasts the value that is be to opened. Here, we can simply buffer the values sent until we have n of them and then do the check in step 3-4 above that P_1 has sent the same values to all players. Note that even if we allow P_1 to send different data to different players for a while, this does not allow information to leak: the fact observed in the simulation proof above, that in any partial opening the honest players always send random independent values, still holds even if P_1 has sent inconsistent data in previous rounds.

A.4 Running the online Phase with Small Fields

Suppose we want error probability $2^{-\mathsf{sec}}$, and $\log p^k$ is much smaller than sec.

When we consider how to solve this problem, we will at first ignore Step 1 in the **Multiply** stage on the online protocol, where one triple is "sacrificed" to check another, as this step could be

done as part of the preprocessing. Nevertheless we do not want to ignore the fact that this step will have a large error probability $1/p^k$. We could solve this by sacrificing $D = \lceil \frac{\mathsf{sec}}{\log p^k} \rceil$ triples instead of one, but we can do much better, and this is described below in Section "A smaller sacrifice" below.

Going back to the actual online phase, we can compensate for the fact that $\log p^k$ is much smaller than sec by setting up the preprocessing so it can work over an extension field K of $\mathbb{F}_{p^k}$ of degree $D = \lceil \frac{\mathsf{sec}}{\log p^k} \rceil$, i.e. an element in K is represented as $\lceil \frac{\mathsf{sec}}{\log p^k} \rceil$ elements from $\mathbb{F}_{p^k}$. All MAC keys and MACs will be generated in K whereas all values to be computed on will still be in $\mathbb{F}_{p^k}$. The preprocessing can ensure this because the ZK proof can already force a prover to choose plaintexts that decode to elements in a subfield of K.

Then error probabilities in the proof of the online phase that were $1/p^k$ before will now be $1/|K| \leq 2^{-\mathsf{sec}}$. The computational complexity of the online phase will now be $O(n|C| + n^3)$ elementary operations in K. Asymptotically, this amounts to $O((n|C| + n^3)D \log D \log\log D)$ elementary operations in $\mathbb{F}_{p^k}$, where the overhead for storage and communication is just D.

It is also possible to get error probability $2^{-\mathsf{sec}}$ while having the preprocessing work only over $\mathbb{F}_{p^k}$. Here the overhead will be larger namely $D^2 \log D \log\log D$, but this may be the best option when D is not very large. The idea is to authenticate by doing D MACs in parallel over $\mathbb{F}_{p^k}$ for every authenticated value, using D independent keys.

We will still do the linear combination $a = \sum_j e_j a_j$ over K, where $e_j = e^j$. This can be done by having the preprocessing generate D random values and thinking of these as an element $e \in K$. Note, however, that we also have to compute a linear combination of the corresponding shares of MACs, i.e., $\gamma_i = \sum_j e_j \gamma(a_j)_i$, and we have D such MACs in parallel. This is why we get a overhead factor $D^2 \log D \log\log D$ for the computational work in this case.

A Smaller Sacrifice. In this section we describe a different method to check the multiplicative relation on triples $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$, where $a, b, c \in \mathbb{F}_{p^k}$. The aim is to decrease the (amortized) number of triples to sacrifice per check. Our approach resembles a technique introduced by Ben-Sasson et al in [4] and one by Cramer et al in [10].

The first step in our construction is to consider a batch of $t+1$ triples $\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle$ for $i = 1, \ldots, t+1$ at once. There are two main ideas in the construction: the first one is to interpolate the values and get polynomials $A, B, C \in \mathbb{F}_{p^k}[X]$ such that $A(i) = a_i$, $B(i) = b_i$, $C(i) = c_i$; if the triples where correctly generated, one would expect $A(x)B(x) = C(x)$ for all x. The second idea is to think of A, B, C as polynomials over a field extension K of $\mathbb{F}_{p^k}$, so that one can check the expected multiplicative relation evaluating A, B, C at a random element $z \in K$; the probability that the check passes even if some of the triples did not satisfy the relation is inversely proportional to the size of K. We now present the full construction.

- Let $\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle$, $i = 1, \ldots, t+1$, be a batch of triples to check.
- One can think of the values $a_1, \ldots, a_{t+1}$ (resp. $b_1, \ldots, b_{t+1}$) as $t+1$ evaluations over $\mathbb{F}_{p^k}$ of a unique polynomial $A \in \mathbb{F}_{p^k}[X]$ (resp. $B \in \mathbb{F}_{p^k}[X]$) of degree t. Concretely, one can define the polynomial A (resp. B) such that $A(i) = a_i$ (resp. $B(i) = b_i$). Since the coefficients of A (resp. B) can be computed as a linear combination of the a_i's (resp. b_i's), the players can compute representations of such coefficients by local computation.
- Players can compute $\langle a_{t+2} \rangle, \ldots, \langle a_{2t+1} \rangle$ such that $A(i) = a_i$, again by local computation, since evaluating a polynomial is a linear operation.

– Players can engage in the multiplication step of the online phase with input $\langle a_i \rangle, \langle b_i \rangle$, and get $\langle c_i \rangle$ (hopefully $c_i = a_i b_i$) for $i = t+2, \ldots, 2t+1$. Notice that players call the multiplication step t times here, so they sacrifice t triples.
– Using only linear computation players can now compute representations of coefficients of the unique polynomial $C \in \mathbb{F}_{p^k}[X]$ of degree $2t$ such that $C(i) = c_i$ for $i = 1, \ldots, 2t+1$.
– Let K be a field extension of $\mathbb{F}_{p^k}$ of degree D. It is possible to think of A, B, C as polynomials over K, by embedding the coefficients via the natural map $\mathbb{F}_{p^k} \longrightarrow K$. Players now evaluate representations for $A(z)B(z)$, and $C(z)$, where z is a public random element in K, and check if $A(z)B(z) = C(z)$ by outputting $A(z)B(z) - C(z)$ and checking if the result is zero. This check can be repeated a number of times in order to lower the error probability. If the check passed all the times, players consider the original triples as valid; otherwise, they discard the triples and start again with fresh triples.

Notice that in order to compute $A(z)B(z)$ and $C(z)$, players need to compute at most D^2 multiplications over $\mathbb{F}_{p^k}$, since $A(z)B(z)$ can be computed by multiplying a $D \times D$ matrix (dependent of $A(z)$) with the vector $B(z)$ (over K, multiplication by a fixed element is an endomorphism of K as a $\mathbb{F}_{p^k}$-vector space). Notice also that we may use the old method of sacrificing more than one triple per multiplication to get any desired error probability for the multiplications over $\mathbb{F}_{p^k}$. We analyze below the error probability we must require.

For the analysis of the construction, one sees that if the multiplicative relation was satisfied by all the original triples, the polynomials AB and C are equal, so the final test passes. In case the triples did not satisfy the relation, then the polynomials AB and C are different, but since they are both of degree at most $2t$, they can agree in at most $2t$ points. Therefore, if z is a root of $AB - C$, then the test passes, and uniform elements in K are roots of $AB - C$ with probability at most $2t/|K|$. If z is not a root of $AB - C$, the test passes only if the multiplication $A(z)B(z)$ does give the correct result, so if we make sure this happens with probability at most $2t/|K|$ (by sacrificing enough triples in the process), then the error probability of the construction is bounded by $2t/|K|$ for a single run of the test. In order to get negligible error probability we reapeat this phase enough times.

An important fact to notice is that in this construction we need $2t + 1 \leq \mathbb{F}_{p^k}$, since otherwise there are not enough elements to evaluate the polynomials. In order to circumvent this restriction, one can still apply the above construction but replacing $\mathbb{F}_{p^k}$ with an extension $\mathbb{F}_{p^{k'}}$ with the required property.

Asymptotically, we see that as we increase the number $t + 1$ of triples checked, we always need to sacrifice t triples, and in addition the number we need to check the multiplication(s) in K. If we assume that we want to hit the desired error probability with just one iteration of the test, we have $2^{-\mathsf{sec}} = 2t/|K|$ from which we get $\log |K| = \mathsf{sec} + \log 2t$. The degree of the extension to K is $\log |K| / \log p^k$, and the number of basic secure multiplications we need is at most the square of this number, which is $(\mathsf{sec} + \log 2t)^2 / (\log p^k)^2$. For each of these, we need error essentially $2^{-\mathsf{sec}}$, so the number of triples we need, say m, satisfies $2^{-\mathsf{sec}} = (1/p^k)^m$, so we get $m = \mathsf{sec} / \log p^k$. This in total grows only poly-logarithmically with t, so we conclude that for a given desired error probability, the number of triples we need to sacrifice to check $t + 1$ triples is $O(t + \mathit{polylog}(t))$.

Comparing the two Approaches: A Concrete Example. We here compare the above approaches for checking triples. Suppose $p = 2$ and $k = 8$, so $\mathbb{F}_{p^k} = \mathbb{F}_{2^8}$. Suppose there are also $t + 1 = 128$ triples to check with security level of 2^{-80}.

Using the latter approach, with $K = \mathbb{F}_{2^{16}}$, we need to sacrifice $t = 127$ triples to generate $\langle c_{t+2}\rangle, \ldots, \langle c_{2t+1}\rangle$; moreover we need to perform 4 secure multiplications to check if $A(z)B(z) = C(z)$, since K is a vector space of dimension 2 over $\mathbb{F}_{2^8}$. In order for the multiplications to be secure enough, we need them to be correct up to error probability $(2 \cdot 127)/2^{16} \approx 2^{-8}$ for the entire multiplication $A(z)B(z)$. This will be the case if for each of the 4 small multiplications we use 3 triples for the multiplication, namely one to do the actual multiplication an two to check the first one. This gives a total error of at most $4 \cdot 2^{-16} \leq 2^{-8}$. So since one run of the test leads to an error probability of $\approx 2^{-8}$, we need 10 runs to decrease the error probability to 2^{-80}. Therefore, the total number of triples to sacrifice is $128 + 4 \cdot 3 \cdot 10 = 248$, while with the original approach the number of triples to sacrifice would have been $128 \cdot 10 = 1280$.

A.5 Preprocessing Phase

Proof (Theorem 3).

Recall first that we assume the cryptosystem has an alternative key generation algorithm $\mathsf{KeyGen}^*()$ which is a randomized algorithm that outputs a *meaningless public key* $\widetilde{\mathsf{pk}}$ with the property 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.

We construct a simulator $\mathcal{S}_{\text{PREP}}$ for Π_{PREP}. In a nutshell, the simulator will run a copy of the protocol. Here, it will play the honest players' part while the environment $\mathcal{Z}$ plays for the corrupt players. The simulator also internally runs copies of $\mathcal{F}_{\text{KeyGen}}$ and $\mathcal{F}_{\text{Rand}}$, in order to simulate calls to these functionalities. Note that in the following we say that the simulator executes or performs some part of the protocol as shorthand for the simulator going through that part with $\mathcal{Z}$. During the protocol execution, whenever $\mathcal{Z}$ sends ciphertexts on behalf of corrupt players, the simulator can obtain the plaintexts, since it knows the secret key. These values are then used to generate input to $\mathcal{F}_{\text{PREP}}$. A precise description is provided in Figure 12.

We now need to show that no $\mathcal{Z}$ can distinguish between the simulated and the real process. By contradiction, we assume that there exists $\mathcal{Z}$ that can distinguish these two cases with significant advantage ϵ. The output of $\mathcal{Z}$ is a single bit, thought of a as guess at one of the two cases. Concretely, we assume

$$\begin{aligned}\mathsf{A}(\mathcal{Z}) := &\Pr\left[\text{“Real”} \leftarrow \mathcal{Z}(\text{Real process})\right] - \Pr\left[\text{“Real”} \leftarrow \mathcal{Z}(\text{Simulated process})\right] \\ &\geq \epsilon.\end{aligned}$$

We will show that such $\mathcal{Z}$ can be used to distinguish between a normally generated public key and a meaningless one with basically the same advantage. This leads to a contradiction, since a key generated by the normal key generator is computationally indistinguishable from a meaningless one.

More in detail, we construct an algorithm B that takes as input a public key pk^* (randomly chosen as either a normal public key or a meaningless one), sets up a copy of $\mathcal{Z}$, goes through the protocol with $\mathcal{Z}$ and uses its output to guess the type of key it got as input. During the process B uniformly chooses a bit (that can be thought as a switch between “Real” and “Simulation”): in case pk^* is correctly computed, if the bit is set to “Real”, $\mathcal{Z}$'s view is indistinguishable from a real execution of the protocol, while if the bit is set to “Simulation”, $\mathcal{Z}$'s view is indistinguishable from a simulated run. However, in case pk^* is meaningless, both choices of the bit lead to statistically

Simulator $\mathcal{S}_{\text{PREP}}$

$\mathsf{SReshare}(e_{\mathbf{m}})$: This is a subroutine the simulator will use while executing the main steps of the protocol described below. Any time in Π_{PREP}, when there is a call to $\mathsf{Reshare}(e_{\mathbf{m}})$, the simulator proceeds as the protocol, but it performs the following extra tasks in order to retrieve the quantity $\Delta_{\mathbf{m}}$:

- On step 2 the simulator decrypts $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{f}_1), \ldots, \mathsf{Enc}_{\mathsf{pk}}(\mathbf{f}_n)$ and obtains the values $\mathbf{f}_1, \ldots, \mathbf{f}_n$
- On step 5 the simulator performs step 2 of $\mathcal{F}_{\text{KeyGenDec}}$, and thereby obtains $\mathbf{m}+\mathbf{f}$ decrypting $e_{\mathbf{m}+\mathbf{f}}$, and $(\mathbf{m}+\mathbf{f})'$ from the adversary
- The simulator sets $\Delta_{\mathbf{m}} \leftarrow (\mathbf{m}+\mathbf{f})' - (\mathbf{m}+\mathbf{f})$, that is $\Delta_{\mathbf{m}}$ is the difference between the output chosen by the adversary for the decryption of $e_{\mathbf{m}+\mathbf{f}}$ and the decryption itself.
- The simulator computes and stores $\mathbf{m}_1 \leftarrow (\mathbf{m}+\mathbf{f})' - \mathbf{f}_1$, and $\mathbf{m}_i \leftarrow -\mathbf{f}_i$ for $i \neq 1$.

Initialize:

- The simulator performs the initialization steps of Π_{PREP}. The call to $\mathcal{F}_{\text{KeyGenDec}}$ in step 1 is simulated by running KeyGen to generate the key pair $(\mathsf{pk}, \mathsf{sk})$. The simulator then sends pk to the players and stores sk.
- Steps 2–5 are performed according to the protocol, but the simulator decrypts every broadcast ciphertext and obtains $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n$
- Step 6 is performed according to the protocol, but the simulator gets $\Delta_1 \leftarrow \mathsf{SReshare}(e_{\gamma(\alpha\cdot\beta_1)}), \ldots, \Delta_n \leftarrow \mathsf{SReshare}(e_{\alpha\cdot\beta_n})$
- The simulator calls Initialize on $\mathcal{F}_{\text{PREP}}$ with input $\{\alpha_i\}_{i\in A}$ at step 1, $\{\beta_i\}_{i\in A}$ at step 3 and $\Delta_1, \ldots, \Delta_n$ at step 5

Pair:

- The simulator performs step 1 according to the protocol
- Steps 2–3 are performed according to the protocol, but the simulator decrypts every broadcast ciphertext and obtains $\mathbf{r}_1, \ldots, \mathbf{r}_n$
- Step 4 is performed according to the protocol, but the simulator gets $\Delta \leftarrow \mathsf{SReshare}(e_{\mathbf{r}\cdot\alpha})$, $\Delta_1 \leftarrow \mathsf{SReshare}(e_{\mathbf{r}\cdot\beta_1}), \ldots, \Delta_n \leftarrow \mathsf{SReshare}(e_{\mathbf{r}\cdot\beta_n})$
- The simulator calls Pair on $\mathcal{F}_{\text{PREP}}$ with input $\{\mathbf{r}_i\}_{i\in A}$ at step 1, and $\Delta, \Delta_1, \ldots, \Delta_n$ at step 3

Triple:

- The simulator performs step 1 according to the protocol
- Steps 2–3 are performed according to the protocol, but the simulator decrypts every broadcast ciphertext and obtains $\mathbf{a}_1, \ldots, \mathbf{a}_n$, $\mathbf{b}_1, \ldots, \mathbf{b}_n$
- Steps 4–5 are performed according to the protocol, but the simulator gets $\Delta_{\mathbf{a}} \leftarrow \mathsf{SReshare}(e_{\mathbf{a}\cdot\alpha})$, $\Delta_{\mathbf{b}} \leftarrow \mathsf{SReshare}(e_{\mathbf{b}\cdot\alpha})$
- Steps 6–7 are performed according to the protocol, but the simulator gets $\mathbf{c}_1, \ldots \mathbf{c}_n$ and $\delta \leftarrow \mathsf{SReshare}(e_{\mathbf{c}})$
- Step 8 is performed according to the protocol, but the simulator gets $\Delta_{\mathbf{c}} \leftarrow \mathsf{SReshare}(e_{\mathbf{c}\cdot\alpha})$
- The simulator calls Triple on $\mathcal{F}_{\text{PREP}}$ with input $\{\mathbf{a}_i\}_{i\in A}$, $\{\mathbf{b}_i\}_{i\in A}$ at step 1, $\Delta_{\mathbf{a}}, \Delta_{\mathbf{b}}, \delta$ at step 3, $\{\mathbf{c}_i\}_{i\in A}$ in step 5, and $\Delta_{\mathbf{c}}$ at step 7

Fig. 12. The simulator for $\mathcal{F}_{\text{PREP}}$.

indistinguishable views. Hence, if $\mathcal{Z}$ guesses correctly whether B chose "Real" or "Simulation", B guesses that pk^* was a standard public key; otherwise B guesses that pk^* was meaningless.

For simplicity we describe the algorithm B for the two-party setting, where there is a corrupt party P_1 and an honest party P_2: On input pk^*, where pk^* is a public key (either meaningless or standard), B starts executing the protocol Π_{PREP}, playing for P_2, while $\mathcal{Z}$ plays for P_1. B does exactly what the simulator would do, with some exceptions:

1. It uses the public key it got as input, instead of generating a key pair initially.
2. B cannot decrypt ciphertexts from P_1 since it does not know the secret key (e.g. at step 4 of Initialize, step 2 of Pair, step 2 of Triple, etc.). Instead, B exploits that P_1 and P_2 ran the protocol Π_{ZKPoPK} with P_1 as prover. That is, P_1 proved that he knows encodings of appropriate size corresponding to the plaintext inside the ciphertexts broadcast in the previous step. This means B can use the knowledge extractor of the protocol Π_{ZKPoPK} followed by decoding to

extract the shares from P_1 (e.g. α_i, β_i at step 4 of Initialize, etc). At this point B continues the protocol as if it had decrypted. Note that the knowledge extractor requires rewinding of the prover (which here effectively is $\mathcal{Z}$). B can do this as it runs its own copy of $\mathcal{Z}$ and since it also controls the copy of $\mathcal{F}_{\text{RAND}}$ used in the protocol, it can issue challenges of its choice to $\mathcal{Z}$.

3. When P_2 gives a ZK proof for a set of ciphertexts, B will simulate the proof. This is done by running the honest verifier simulator to get a transcript $(\mathbf{a}, \mathbf{e}, (\mathbf{z}, T))$ and letting the copy of $\mathcal{F}_{\text{RAND}}$ output $\mathbf{e}$ that occurs in the simulate transcript.

In the end B uniformly chooses to generate a real or a simulated view. In the first case, B outputs to $\mathcal{Z}$ exactly those values for P_2 that were used in the execution of the protocol. In the other case, B generates the output for P_2 as $\mathcal{F}_{\text{PREP}}$ would do. That means that P_2's shares $\mathbf{a}_2, \mathbf{b}_2, \mathbf{c}_2$ of a triple $\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle, \langle \mathbf{c} \rangle$ will be determined by choosing $\mathbf{a}, \mathbf{b}$ at random, setting $\mathbf{c} \leftarrow \mathbf{a} \cdot \mathbf{b}$ and then letting $\mathbf{a}_2 \leftarrow \mathbf{a} - \mathbf{a}_1^{Real}$, $\mathbf{b}_2 \leftarrow \mathbf{b} - \mathbf{b}_1^{Real}$, $\mathbf{c}_2 \leftarrow \mathbf{c} - \mathbf{c}_1^{Real}$.

It can now be seen that if pk^* is a normal key, then the view generated by B corresponds statistically to either a real or a simulated execution: if B chooses the simulation case, the only differences to the actual simulator are 1) the simulator executes the ZK proofs given by P_2 according to the protocol while B simulates them; and 2) the simulator opens the ciphertexts using the secret key to decrypt, while B uses the extractor for Π_{ZKPoPK} and computes the plaintexts from its results. As for 1) the ZK proof is statistical ZK so this leads to a statistically indistinguishable distribution. As for 2), note that for every ciphertext $e_{\mathbf{x}}$ generated by P_1, the extractor for Π_{ZKPoPK} will, except with negligible probability, be able to find an encoding $\mathbf{x}$ (resp. randomness r) smaller than B_{plain} (resp. B_{rand}), with $e_{\mathbf{x}} = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}, \mathbf{r})$. This follows from soundness of Π_{ZKPoPK} and admissibility of the cryptosystem. Then, by correctness of the cryptosystem, computing the plaintexts as B does, will indeed give the same result as decrypting, except with negligible probability. If B chooses the real case, a similar argument shows that we get a view statistically indistinguishable from a real run of the protocol. Hence if pk^* is a normal key, $\mathcal{Z}$ can guess B's choice of "Real" or "Simulation" with advantage essentially ϵ.

On the other hand if pk^* is a meaningless key, the encryptions contain statistically no information about the values inside. Moreover, all messages sent in the zero-knowledge protocols where P_2 acts as prover, do not depend on the specific values that P_2 has, since the proofs are simulated. We conclude that essentially no information on any value held by P_2 is revealed. This is the case also for step 5 of $\mathsf{Reshare}(e_{\mathbf{m}})$: $\mathbf{m}+\mathbf{f}$ is retrieved, but no information on $\mathbf{m}$ is revealed, since $\mathbf{f}$ is uniform.

The view $\mathcal{Z}$ sees consists of the view of the corrupt player(s) and the output of the honest player(s). We just argued that the view of the corrupt player is essentially independent of the internal values B uses for P_2, and hence also independent of whether B chooses the real or the simulated case. Therefore, the output generated for the honest player(s) seen by $\mathcal{Z}$ is in both cases a set of (essentially) uniformly and independently chosen shares and MAC keys. As a result, if we use a meaningless key, a real execution and a simulated execution are statistically indistinguishable, and the guess of $\mathcal{Z}$ will equal B's random choice of "Real" or "Simulation" with probability essentially $1/2$.

An easy calculation now shows that the advantage of B is

$$\begin{aligned}\mathsf{A}(B) &:= \Pr\left[\text{"Standard Key"} \leftarrow B(\mathsf{pk})\right] - \Pr\left[\text{"Standard Key"} \leftarrow B(\widetilde{\mathsf{pk}})\right] \\ &\geq \mathsf{A}(\mathcal{Z})/2 - \delta \\ &= \epsilon/2 - \delta,\end{aligned}$$

for some negligible δ that accounts for the differences between the involved distributions. However, if ϵ is non-negligible, then $\epsilon/2 - \delta$ is also non-negligible, which contradicts the assumption on that meaningless keys are statistically indistinguishable from standard ones. □

A.6 Distributed Decryption

Proof (Theorem 4). The requirement $B + 2^{\mathsf{sec}} \cdot B < q/2$ implies that $\mathbf{t}' = \mathbf{t} \bmod p$, since $\|\mathbf{r}_i\|_\infty < 2^{\mathsf{sec}} \cdot B/(n \cdot p)$ for $i = 1, \ldots, n$. Therefore the protocol allows players to retrieve the correct message if all the players are honest.

We now build a simulator $\mathcal{S}_{\text{DDEC}}$ to work on top of $\mathcal{F}_{\text{KEYGENDEC}}$, such that the adversary cannot distinguish whether it is playing with the decryption protocol and $\mathcal{F}_{\text{KEYGEN}}$ or the simulator and $\mathcal{F}_{\text{KEYGENDEC}}$. We let A denote the set of players controlled by the adversary.

Simulator $\mathcal{S}_{\text{DDEC}}$

Key Generation: This stage is needed to distribute shares of a secret key.
- Upon "start", the simulator sends "start" to $\mathcal{F}_{\text{KEYGENDEC}}$ and obtains pk. Moreover, the simulator obtains $(\mathsf{sk}_i)_{i \in A}$ from the adversary.
- The simulator (internally) sets random $(\mathsf{sk}_i)_{i \notin A}$ such that $(\mathsf{sk}_i)_{i=1,\ldots,n}$ is a full vector of shares of 0.
- The simulator sends pk to A.

Public Decryption: This stage simulates a public decryption.
- Upon "decrypt c, B", the simulator sends "decrypt c" to $\mathcal{F}_{\text{KEYGENDEC}}$ and obtains $m = \mathsf{Dec}_{\mathsf{sk}}(c)$.
- It then computes the value $\mathbf{v}_i$ for all players except for an honest player P_j.
- It then samples $\mathbf{r}_j$ uniformly with infinity norm bounded by $2^{\mathsf{sec}} \cdot B/(n \cdot p)$ and computes

$$\widetilde{\mathbf{t}}_j \leftarrow -\sum_{i \neq j} \mathbf{v}_i + p \cdot \mathbf{r}_j + \mathsf{encode}(m).$$

- For each other honest player P_i, it computes $\mathbf{t}_i$ honestly (using c, sk_i).
- The simulator broadcasts the values $(\mathbf{t}_i)_{i \notin A, i \neq j}, \widetilde{\mathbf{t}}_j$ and obtains $(\mathbf{t}_i^*)_{i \in A}$ from the adversary.
- It then sends $m' \leftarrow \mathsf{decode}\left(\left(\widetilde{\mathbf{t}}_j + \sum_{i \in A} \mathbf{t}_i^* + \sum_{i \notin A, i \neq j} \mathbf{t}_i\right) \bmod p\right)$ to $\mathcal{F}_{\text{KEYGENDEC}}$ so that the ideal functionality sends "Result m'" to all the players.

Private Decryption: This stage simulates a private decryption.
- Upon "decrypt c, B to P_j", the simulator sends "decrypt c to P_j" to $\mathcal{F}_{\text{KEYGENDEC}}$.
- If P_j is corrupt, the simulator obtains $c, m = \mathsf{Dec}_{\mathsf{sk}}(c)$ from $\mathcal{F}_{\text{KEYGENDEC}}$ and acts as in the simulated public decryption.
- If P_j is honest, the simulator receives c from $\mathcal{F}_{\text{KEYGENDEC}}$, $\mathbf{t}_i^*$ from each corrupt player P_i and $\mathbf{t}_i$ from each honest player.
  - The simulator samples $\mathbf{r}_j$ uniformly with infinity norm bounded by $2^{\mathsf{sec}} \cdot B/(n \cdot p)$.
  - It evaluates $\widetilde{\mathbf{t}}_j \leftarrow -\sum_{i \neq j} \mathbf{v}_i + p \cdot \mathbf{r}_j$.
  - It computes $\varepsilon \leftarrow \left(\widetilde{\mathbf{t}}_j + \sum_{i \in A} \mathbf{t}_i^* + \sum_{i \notin A, i \neq j} \mathbf{t}_i\right) \bmod p$
  - Finally it sends $\delta \leftarrow \mathsf{decode}(\varepsilon)$ to $\mathcal{F}_{\text{KEYGENDEC}}$ in order to get $\mathsf{Dec}_{\mathsf{sk}}(c) + \delta$ to P_j.

Fig. 13. The simulator for Π_{DDEC}.

In a simulated decryption the adversary receives pk and $(\mathbf{t}_i)_{i \notin A, i \neq j}, \widetilde{\mathbf{t}}_j$ from $\mathcal{S}_{\text{DDEC}}$. The distribution of pk is the same as in a real conversation, since it was sampled using the same algorithm as in a real conversation. The distribution of simulated $\mathbf{t}_i$, $i \neq j$ is statistically close to the real one, since $\mathbf{t}_i$ was computed correctly using shares of a possible secret key. We can therefore focus on the case where all the players but one are dishonest. We first analyse the simulation of public

decryption, introducing a hybrid machine, and prove its output is statistically indistinguishable from P_j's output (in the real protocol) and perfectly indistinguishable from P_j's simulated output.

Hybrid: On input $(\mathsf{sk}_i)_{i=1,\dots,n}, c$, reconstruct sk, compute $\mathsf{Dec}_{\mathsf{sk}}(c)$, sample $\mathbf{r}_j$ uniformly with infinity norm bounded by $2^{\mathsf{sec}} \cdot B/(n \cdot p)$ and output $\widetilde{\mathbf{t}}_j \leftarrow -\sum_{i \neq j} \mathbf{v}_i + p \cdot \mathbf{r}_j + \mathsf{encode}(m)$.

Notice that $\widetilde{\mathbf{t}}_j = \mathbf{v}_j - \mathbf{t} + \mathsf{encode}(m) + p \cdot \mathbf{r}_j$. Now, for a distribution X, define $\varphi(X) := p \cdot X + \mathbf{v}_j$. Notice that $\mathbf{t}_j = \varphi(U)$, where U denotes the uniform distribution over vectors of integral entries bounded with infinity norm $2^{\mathsf{sec}} \cdot B/(n \cdot p)$; moreover, since $\mathbf{t} - \mathsf{encode}(m)$ is a multiple of p, one can write $\widetilde{\mathbf{t}}_j = \varphi(U + (\mathsf{encode}(m) - \mathbf{t})/p)$. Notice that $\|(\mathsf{encode}(m) - \mathbf{t})/p\|_\infty \leq (B+p)/p$, so the distribution $U + (\mathsf{encode}(m) - \mathbf{t})/p$ is statistically close to U, since the probability of distinguishing $U + (\mathsf{encode}(m) - \mathbf{t})/p$ and U is bounded by the ratio

$$\frac{\|(\mathsf{encode}(m) - \mathbf{t})/p\|_\infty}{2^{\mathsf{sec}} \cdot B/(n \cdot p) + (B+p)/p} \leq \frac{(B+p)/p}{2^{\mathsf{sec}} \cdot B/(n \cdot p) + (B+p)/p} = O(n \cdot 2^{-\mathsf{sec}}),$$

which is negligible. Therefore $\widetilde{\mathbf{t}}_j$ is statistically close to $\mathbf{t}_j$.

What is left to prove is that the simulation of private decryption to an honest player P_j is statistically indistinguishable from the real protocol. In the real protocol P_j computes $\mathbf{t}_j$ and

$$m' \leftarrow \mathsf{decode}\left(\sum_{i \in A} \mathbf{t}_i^* + \sum_{i \notin A} \mathbf{t}_i\right).$$

In that case the error $m' - m$ introduced by the adversary depends only on the value

$$\varepsilon' := \left(\sum_{i \in A} (\mathbf{t}_i^* - \mathbf{t}_i)\right) \bmod p$$

computed using the actual secret key. In the simulation the error introduced by the adversary is

$$\varepsilon = \left(\widetilde{\mathbf{t}}_j + \sum_{i \in A} \mathbf{t}_i^* + \sum_{i \notin A, i \neq j} \mathbf{t}_i\right) \bmod p = \left(\sum_{i \in A} (\mathbf{t}_i^* - \mathbf{t}_i)\right) \bmod p,$$

computed using secret shares of 0. Since the secret sharing scheme has privacy threshold n and the sums involve at most $n-1$ shares, the quantities ε and ε' are statistically indistinguishable. □

B A lower Bound for the Preprocessing

In this section, we show that any preprocessing matching the properties we have, must output the same amount of data as we do, up to a constant factor. We use the following theorem for 2-party computation from [28]. It talks about a setting where the parties A, B have access to a functionality that gives a random variable U to A and V to B with some guaranteed joint distribution P_{UV} of U, V. Given this, the parties compute securely a function $f : \mathcal{X} \times \mathcal{Y} \mapsto \mathcal{Z}$, where A holds $x \in \mathcal{X}$, and B holds $y \in \mathcal{Y}$. This function should have the property that there exists inputs y_1, y_2 such that for all $x \neq x'$, $f(x, y_1) \neq f(x', y_1)$; and for all x, x', $f(x, y_2) = f(x', y_2)$. In other words, for some inputs B learns all of A's input, but other inputs B learns nothing new.

Theorem 6. *Let $f : \mathcal{X} \times \mathcal{Y} \mapsto \mathcal{Z}$ be a function with inputs y_1, y_2 as above. If there exists a protocol that computes f securely with access to P_{UV} and with error probability ϵ in the semi-honest model, then*

$$H(V) \geq I(U;V) \geq \log|\mathcal{X}| - 7(\epsilon \log|\mathcal{X}| + h(\epsilon))$$

We will also need the following technical lemma

Lemma 1. *Let R be a random variable defined over the natural numbers. Then there exists a constant C such that $E(R) \geq H(R) - 1 - C$.*

Proof (Lemma 1). Let

$$I := \left\{ i \mid i \geq \log\left(\frac{1}{Pr[R=i]}\right) \right\}.$$

Under such a definition, one can write $H(R)$ as

$$H(R) = \sum_{i \in I} Pr[R=i] \cdot \log\left(\frac{1}{Pr[R=i]}\right) + \sum_{i \notin I} Pr[R=i] \cdot \log\left(\frac{1}{Pr[R=i]}\right)$$

By the construction of I, one can bound the first summand as follows

$$\begin{aligned} \sum_{i \in I} Pr[R=i] \cdot \log\left(\frac{1}{Pr[R=i]}\right) &\leq \sum_{i \in I} Pr[R=i] \cdot i \\ &\leq \sum_{i} Pr[R=i] \cdot i \\ &= E(R). \end{aligned}$$

For the second summand one needs to work a bit more. Let $q(i) := \log(1/Pr[R=i])$. Then

$$\sum_{i \notin I} Pr[R=i] \cdot \log\left(\frac{1}{Pr[R=i]}\right) = \sum_{i \notin I} 2^{-q(i)} \cdot q(i).$$

We now claim that

$$2^{-q(i)} \cdot q(i) \leq 2^i \cdot i, \text{ for all } 0 \neq i \notin I.$$

This happens if and only if

$$2^{-q(i)} \cdot 2^{\log(q(i))} \leq 2^{-i} \cdot 2^{\log(i)}.$$

Taking the logarithm of such relation one gets $-q(i) + \log(q(i)) \leq -i + \log(i)$, which is equivalent to $q(i) - \log(q(i)) \geq i - \log(i)$.
Since $q(i) = \log(1/Pr[R=i]) \geq i$ for all $i \notin I$, and $i \geq 1$, the latter relation is always satisfied. Therefore, one can bound the second summand by $C + \sum_{i \geq 1} 2^{-i} \cdot i$, where $C = 2^{-q(0)} \cdot q(0)$.
Moreover $\sum_{i \geq 1} 2^{-i} \cdot i$ converges to 1, so the second summand can be bound by $1 + C$.

Finally, one can reassemble all the reasoning into one and get

$$\sum_{i \in I} Pr[R=i] \cdot \log\left(\frac{1}{Pr[R=i]}\right) + \sum_{i \notin I} Pr[R=i] \cdot \log\left(\frac{1}{Pr[R=i]}\right) \leq E(R) + C + 1.$$

The last inequality implies that $H(R) \leq E(R) + 1 + C$ □

With this result, we can prove the lower bound claimed earlier:

Proof (Theorem 2). Suppose we have an on-line protocol π that satisfies the assumptions in the theorem. Consider any player P_i and suppose we want to compute the function

$$f_T((\boldsymbol{x}, \boldsymbol{x}'), y) = y\boldsymbol{x} + (1-y)\boldsymbol{x}'.$$

Here $y \in \mathbb{F}_{p^k}$ and $\boldsymbol{x}, \boldsymbol{x}'$ are vectors over $\mathbb{F}_{p^k}$ of length T. P_i will have input y and each P_j, $j \neq i$ will have as input substrings $\boldsymbol{x}_j, \boldsymbol{x}'_j$ such that the concatenation of all $\boldsymbol{x}_j$ ($\boldsymbol{x}'_j$) is $\boldsymbol{x}$ ($\boldsymbol{x}'$). Finally, only P_i learns the output $f_T((\boldsymbol{x}, \boldsymbol{x}'), y)$.

Clearly, f_T can be computed using a circuit of size $O(T)$, and this will be the circuit promised in the theorem. Note that our assumed protocol π can handle circuits of size S and can therefore compute f_T securely where T is $\Theta(S)$.

We can now transform π to a two-party protocol π' for parties A and B. A has input $\boldsymbol{x}, \boldsymbol{x}'$, B has input y and B is supposed to learn $f_T((\boldsymbol{x}, \boldsymbol{x}'), y)$. Now, π' simply consists of running π where B emulates P_i and A emulates all other players. We give to B whatever P_i gets from the preprocessing and A gets whatever the other players receive, so this defines the random variables U and V. Since π is secure if P_i is corrupt and also if all other players are corrupt, this trivially means that π' is an actively secure two-party protocol for computing f_T.

This implies that π' also computes f_T with passive security. As noted in [28], this is actually not necessarily the case for all functions. The problem is that if the adversary is passive, then active security does guarantees that there is a simulator for this case, but such a simulator is allowed to change the inputs of corrupted parties. A simulator for the passive case is not allowed to do this. However, [28] observe that for some functions, an active simulator cannot get away with changing the inputs, as this would make it impossible to simulate correctly. They show this is the case for Oblivious Transfer which is essentially what f_T is after we go to the 2-party case. We may therefore assume π' is also passively secure.

Finally, we define $f'_T(\boldsymbol{x}, y) = f_T((\boldsymbol{x}, \boldsymbol{0}), y) = y\boldsymbol{x}$. Obviously π' can be used to compute f'_T securely, A just sets her second input to be $\boldsymbol{0}$. Moreover f'_T satisfies the conditions in Theorem 6. So we get that $H(V) \geq \log|\mathcal{X}| - 7(\epsilon \log|\mathcal{X}| + h(\epsilon))$. If we adopt the standard convention that the security parameter grows linearly with the input size $\log|\mathcal{X}|$ then because ϵ is negligible in the security parameter, we have that the "error term" $7(\epsilon \log|\mathcal{X}| + h(\epsilon))$ is $o(\log|\mathcal{X}|)$.

So we get that $H(V)$ is $\Omega(\log|\mathcal{X}|) = \Omega(T \log p^k) = \Omega(S \log p^k)$, since T is $\Theta(S)$. Recalling that $H(V)$ is actually the entropy of the variable P_i received in the original protocol π, we get the first conclusion of the Theorem.

For the second conclusion about the computational work done, it is tempting to simply claim that B has to at least read the information he is given and so $H(V)$ is a lower bound on the expected number of bit operations. But this is not enough. It is conceivable that in every particular execution, B might only have to read a small part of the information.

It turns out that this does not happen, however, which can be argued as follows: let $B(V)$ be the random variable representing the bits of V that B actually reads. By inspection of the proof of Theorem 6, one sees that if we replace everywhere V by $B(V)$ the same proof still applies. So in fact, we have $H(B(V)) \geq \log|\mathcal{X}| - 7(\epsilon \log|\mathcal{X}| + h(\epsilon))$. Now let R be the random variable representing the number of bits B reads from V.

If we condition on R, then the entropy of $B(V)$ cannot drop by more than $H(R)$, so we have

$$H(B(V)|R) \geq H(B(V)) - H(R) \geq \log|\mathcal{X}| - 7(\epsilon \log|\mathcal{X}| + h(\epsilon)) - H(R).$$

Moreover, we also have

$$H(B(V)|R) = \sum_r Pr(R=r)H(B(V)|R=r) \leq \sum_r Pr(R=r)r = E(R)$$

Putting these two inequalities together, we obtain that

$$E(R) + H(R) \geq \log|\mathcal{X}| - 7(\epsilon \log|\mathcal{X}| + h(\epsilon)).$$

Now, either $E(R) \geq (\log|\mathcal{X}| - 7(\epsilon\log|\mathcal{X}| + h(\epsilon)))/2$, or $H(R) \geq (\log|\mathcal{X}| - 7(\epsilon\log|\mathcal{X}| + h(\epsilon)))/2$. In the latter case we have from Lemma 1 that $E(R)$ is much larger than $H(R)$, so we can certainly conclude that $E(R) \geq (\log|\mathcal{X}| - 7(\epsilon\log|\mathcal{X}| + h(\epsilon)))/2$ in any case. As above, the error term depending on ϵ becomes negligible for increasing security parameter, so we get that $E(R)$ is $\Omega(S\log p^k)$ as desired. □

C Canonical Embeddings of Cyclotomic Fields

Our concrete instantiation will use some basic results of Cyclotomic fields which we now recap on; these results are needed for the main result of this Appendix which is a proof of a "folklore" result about the relationship between norms in the canonical and polynomial embeddings of a cyclotomic field. This result is used repeatedly in our main construction to produce estimates on the size of parameters needed.

C.1 Cyclotomic Fields

We first recap on some basic facts about numbers fields, and their canonical embeddings. Focusing particularly on the case of cyclotomic fields.

Number Fields An algebraic number (resp. algebraic integer) $\theta \in \mathbb{C}$ is the root of a polynomial (resp. monic polynomial) with coefficients in $\mathbb{Q}$ (resp. $\mathbb{Z}$). The minimal polynomial of θ is the unique monic irreducible $f(x) \in \mathbb{Q}[X]$ which has θ as a root.

A number field $K = \mathbb{Q}(\theta)$ is the field obtained by adjoining powers of an algebraic number θ to $\mathbb{Q}$. If θ has minimal polynomial $f(x)$ of degree N, then K can be considered as a vector space over $\mathbb{Q}$, of dimension N, with basis $\{1, \theta, \ldots, \theta^{N-1}\}$. Note that this "coefficient embedding" is relative to the defining polynomial $f(x)$ Equivalently we have $K \cong \mathbb{Q}[X]/f(X)$, i.e. the field of rational polynomials with degree less than N, modulo the polynomial $f(X)$. Without loss of generality we can assume K, from now on, is defined by a monic irreducible integral polynomial of degree N. The ring of integers $\mathcal{O}_K$ of K is defined to be the subring of K consisting of all elements whose minimal polynomial has integer coefficients.

Canonical Embedding There are N field morphisms $\sigma_i : K \longrightarrow \mathbb{C}$ which fix every element of $\mathbb{Q}$. Such a morphism is called a complex *embedding* and it takes θ to each distinct complex root of $f(X)$. The number field K is said to have signature (s_1, s_2) if the defining polynomial has s_1 real roots and s_2 complex conjugate pairs of roots; clearly $N = s_1 + 2 \cdot s_2$. The roots are numbered in the standard way so that $\sigma_i(\theta) \in \mathbb{R}$ for $1 \leq i \leq s_1$ and $\sigma_{i+s_1+s_2}(\theta) = \overline{\sigma_{i+s_1}(\theta)}$ for $1 \leq i \leq s_2$. We define $\sigma = (\sigma_1, \ldots, \sigma_N)$, which defines the *canonical embedding* of K into $\mathbb{R}^{s_1} \times \mathbb{C}^{2\cdot s_2}$, where the field operations in K are mapped into componentwise addition and multiplication in $\mathbb{R}^{s_1} \times \mathbb{C}^{2\cdot s_2}$. To ease notation we will often write $\alpha^{(i)} = \sigma_i(\alpha)$, for $\alpha \in K$. We will let $\|\alpha\|_p$ for $p \in [1, \ldots, \infty]$ denote the p-norm of α in the coefficient embedding (i.e. the p-norm of the vector of coefficients) and let $\|\sigma(\alpha)\|_p$ denote norms in the canonical embedding.

Cyclotomic Fields We will mainly be concerned with cyclotomic number fields. The mth cyclotomic polynomial is given by $\Phi_m(X)$, this is an irreducible polynomial of degree $N = \phi(m)$. The number field defined by $\Phi_m(X)$ is said to be a cyclotomic number field, and is defined by $K = \mathbb{Q}(\zeta_m)$, where ζ_m is an mth root of unity, i.e. a root of $\Phi_m(X)$. The ring of integers of K is equal to $\mathbb{Z}[\zeta_m]$. The number field K is Galois, and hence (importantly for us) the polynomial splits modulo p (for any prime p not dividing m) into a produce of distinct irreducible polynomials all of the same degree.

The key fact is that if $\Phi_m(X)$ has degree d factors modulo the prime p then m divides $p^d - 1$. To see this notice that if $\Phi_m(X)$ factors into N/d factors each of degree d then the finite field $\mathbb{F}_{p^d}$ must contain the mth roots of unity and so m divides $p^d - 1$. In the other direction, if d is the smallest integer such that m divides $p^d - 1$ then $\Phi_m(X)$ will have a degree d factor since the decomposition group of the prime p in the Galois group will have order d.

C.2 Relating Norms Between Canonical and Polynomial Embeddings

There is a distinct difference between the canonical and polynomial embeddings of a number field. In particular notice the following expansions upon multiplication, for $x, y \in \mathcal{O}_K$,

$$\|x \cdot y\|_\infty \leq \delta_\infty \cdot \|x\|_\infty \cdot \|y\|_\infty.$$
$$\|\sigma(x \cdot y)\|_p \leq \|\sigma(x)\|_\infty \cdot \|\sigma(y)\|_p.$$

where

$$\delta_\infty = \sup \left\{ \frac{\|a(X) \cdot b(X) \pmod{f(X)}\|_\infty}{\|a(X)\|_\infty \cdot \|b(X)\|_\infty} : a, b \in \mathbb{Z}[X], \deg(a), \deg(b) < N \right\}.$$

In this section we show that one can more tightly control the expansion factor of elements in the polynomial representation; as long as they are drawn randomly with a discrete Gaussian distribution. In particular we prove the following theorem; this result is well known to people working in ideal lattice theory, but proofs have not yet appeared in any paper.

Theorem 7. *Let K denote a cyclotomic number field then there is a constant C_m, depending only on m, such that for all $\alpha \in \mathcal{O}_K$ we have*

- $\|\sigma(\alpha)\|_\infty \leq \|\alpha\|_1$.
- $\|\alpha\|_\infty \leq C_m \cdot \|\sigma(\alpha)\|_\infty$.

We recall some facts about various matrices associated with roots of unity, see [26] and the full version of [22]. First some notation; for any integer $m \geq 2$: We set $\zeta_m = \exp(2 \cdot \pi \cdot \sqrt{-1}/m)$ to be a root of unity for an integer m. As usual we let $N = \phi(m)$ and we define $\mathbb{Z}_m^* = \{a_{m,i} : 0 \leq i < N\}$ to be a complete set of representatives for $\mathbb{Z}_m^*$ with $1 \leq a_{m,i} < m$. We let $A \otimes B$, for matrices A and B, denote the Kronecker product. We let I_t denote the $t \times t$ identity matrix. All $a \times b$ matrices M in this section will have elements $m_{i,j}$ indexed by $0 \leq i < a$ and $0 \leq j < b$; i.e. we index from zero; this is to make some of the expressions easier to write down. The infinity norm for a matrix $M = (m_{i,j})$ is defined by

$$\|M\|_\infty := \max \left\{ \sum_{j=0}^{N-1} |m_{i,j}| \right\}_{i=0}^{N-1}.$$

We define the $N \times N$ CRT matrix as follows:

$$\mathsf{CRT}_m := \left(\zeta_m^{a_{m,i} \cdot j}\right)_{0 \leq i,j < N}.$$

Then we define the constant C_m in the above theorem as $C_m = \|\mathsf{CRT}_m^{-1}\|_\infty$. From which the proof now immediately follows:

Proof (Theorem 7). For a cyclotomic field the canonical embedding is given by the map $\sigma(\alpha) = \mathsf{CRT}_m \cdot \alpha$, where α is the vector of the coefficient embedding of α, i.e. α considered as a polynomial in θ a root of $F(X) = \Phi_m(X)$ and CRT_m is the matrix, defined earlier, i.e. it is equal to

$$\mathsf{CRT}_m = \begin{pmatrix} 1 & \theta^{(1)} & \dots & \theta^{(1)N-1} \\ \vdots & \vdots & & \vdots \\ 1 & \theta^{(N)} & \dots & \theta^{(N)N-1} \end{pmatrix}.$$

For the first part of the theorem we note that, on writing $\alpha = \sum_{i=0}^{N-1} x_i \cdot \theta^i$, we have

$$\left|\alpha^{(i)}\right| = \left|\sum_{j=0}^{N-1} x_j \cdot \theta^{(i)j}\right| \leq \sum_{j=0}^{N-1} |x_j| \cdot |\theta^{(i)j}| = \sum_{j=0}^{N-1} |x_j| = \|\mathbf{x}\|_1 = \|\alpha\|_1.$$

For the second part we note that for all $\beta \in \mathcal{O}_K$,

$$\|\beta\|_\infty = \|\mathsf{CRT}_m^{-1} \cdot \sigma(\beta)\|_\infty \leq \|\mathsf{CRT}_m^{-1}\|_\infty \cdot \|\sigma(\beta)\|_\infty$$

from which the result follows. □

The key question then is how large can C_m become. So we now turn to this problem; giving a partial answer.

The $m \times m$ DFT matrix is defined by:

$$\mathsf{DFT}_m := \left(\zeta_m^{i \cdot j}\right)_{0 \leq i,j < m}.$$

Let m' be a divisor of m then for $i \in \{0, \dots, m-1\}$ we write $i_0 = i \bmod m'$ and $i_1 = (i - i_0)/m'$. We then define the $m \times m$ "twiddle matrix" to be the diagonal matrix defined by

$$T_{m,m'} := \mathsf{Diag}\left\{\zeta_m^{i_0 \cdot i_1}\right\}_{i=0,\dots,m-1}.$$

Finally we define $L_{m'}^m$ to be the permutation matrix which fixes the row with index $m-1$, but sends all other rows i, for $0 \leq i < m-1$, to row $i \cdot m' \bmod m - 1$. Following [26] we use these matrices to decompose the matrix D_m into D'_m and D_k, where $m = m' \cdot k$, via the following identity

$$\mathsf{DFT}_m = L_{m'}^m \cdot (I_k \otimes \mathsf{DFT}_{m'}) \cdot T_{m,m'} \cdot (\mathsf{DFT}_k \otimes I_{m'}), \tag{3}$$

This is nothing but the general Cooley-Tukey decomposition of the DFT for composite m. Consider the Vandermonde matrix

$$V(x_1, \dots, x_m) := \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{m-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^{m-1} \end{pmatrix}.$$

It is clear that $\mathsf{DFT}_m = V(1, \zeta_m, \zeta_m^2, \dots, \zeta_m^{m-1})$.

Lemma 2. *We have, for any m,*

$$\mathsf{DFT}_m^{-1} = \frac{1}{m} \cdot V(1, \zeta_m^{-1}, \zeta_m^{-2}, \ldots, \zeta_m^{1-m})$$

Proof. Let $\delta_{i,j}$ be defined so that $\delta_{i,j} = 0$ if $i \neq j$ and equal to one otherwise. We have

$$\begin{aligned}\left(V(1, \zeta_m, \zeta_m^2, \ldots, \zeta_m^{m-1}) \cdot V(1, \zeta_m^{-1}, \zeta_m^{-2}, \ldots, \zeta_m^{1-m})\right)_{i,j} &= \sum_{0 \leq k < m} \zeta_n^{i \cdot k} \cdot \zeta_m^{-k \cdot j} \\ &= \sum_{0 \leq k < m} \zeta_m^{k \cdot (i-j)} = m \cdot \delta_{i,j}.\end{aligned}$$

□

This leads to the following lemma which gives shows that the infinity norm of the inverse of the DFT matrix is always equal to one.

Lemma 3. *For any m we have $\|\mathsf{DFT}_m^{-1}\|_\infty = 1$.*

Proof. If ζ_m is an m-th root of unity, it is clear that $\|V(1, \zeta_m, \zeta_m^2, \ldots, \zeta_m^{m-1})\|_\infty = m$. In addition we have $\psi_m = 1/\zeta_m$ is also an m-th root of unity, thus

$$\|\mathsf{DFT}_m^{-1}\|_\infty = \frac{1}{m} \cdot \|V(1, \psi_m, \psi_m^2, \ldots, \psi_m^{m-1})\|_\infty = \frac{m}{m} = 1.$$

□

Let $m = p_1^{e_1} \cdots p_k^{e_s}$ we define $r = p_1 \cdots p_s$, $m_1 = m/r$; hence $N = \phi(m) = \phi(r) \cdot m_1$. In [22] the authors specialise the decomposition (3) (by selecting appropriate rows and columns) in the case $m' = m_1$ and $k = r$, to show that to show that, upto a permutation of the rows, the matrix CRT_m is equal to

$$\left(I_{\phi(r)} \otimes \mathsf{DFT}_{m_1}\right) \cdot T^*_{m,m_1} \cdot (\mathsf{CRT}_r \otimes I_{m_1})$$

where ${T_{m,m_1}}^*$ is another diagonal matrix consisting of roots of unity. We then have that

Lemma 4. *For an integer $m \geq 2$ such that $m = p_1^{e_1} \cdots p_k^{e_k}$ we write $r = p_1 \cdots p_k$, we then have $C_m \leq C_r$.*

Proof. As above we write $m_1 = m/r$. First note that $\|A \otimes I_t\|_\infty = \|I_s \otimes A\|_\infty = \|A\|_\infty$ for any matrix A and any integers s and t. Then also note that since CRT_m is given, upto a permutation of the rows, by the above decomposition, we have that CRT_m^{-1} is given up to a permutation of the rows by the decomposition

$$\left(\mathsf{CRT}_r^{-1} \otimes I_{m_1}\right) \cdot T^{-1} \cdot \left(I_{\phi(r)} \otimes \mathsf{DFT}_{m_1}^{-1}\right).$$

So we have

$$\begin{aligned}\|\mathsf{CRT}_m^{-1}\|_\infty &= \| \left(\mathsf{CRT}_r^{-1} \otimes I_{m_1}\right) \cdot T^{-1} \cdot \left(I_{\phi(r)} \otimes \mathsf{DFT}_{m_1}^{-1}\right) \|_\infty, \\ &\leq \|\mathsf{CRT}_r^{-1} \otimes I_{m_1}\|_\infty \cdot \|T^{-1}\|_\infty \cdot \|I_{\phi(r)} \otimes \mathsf{DFT}_{m_1}^{-1}\|_\infty, \\ &= \|\mathsf{CRT}_r^{-1}\|_\infty \cdot \|T^{-1}\|_\infty \cdot \|\mathsf{DFT}_{m_1}^{-1}\|_\infty = \|\mathsf{CRT}_r^{-1}\|_\infty.\end{aligned}$$

□

This result means that we can bound C_m for infinite families of values of m, by simply deducing a bound on C_r, where r is the product of all primes dividing m. For example notice that $\mathsf{CRT}_r = (1)$ and hence $C_{2^e} = C_2 = 1$ for all values of e. Indeed it is relatively straight forward to determine the exact value of C_p for a prime p:

Lemma 5. *If p is a prime then*

$$C_p = \frac{2 \cdot \sin(\pi/p)}{p \cdot (\cos(\pi/p) - 1)}.$$

Proof. [7] First note that it is a standard fact from algebra (by consider inverses of Vandermonde matrices for example) that the entries of a row of the matrix CRT_p^{-1} are given by the coefficients of the polynomial

$$\frac{\Phi_p(X)}{\Phi_p'(\zeta_p) \cdot (X - \zeta_p)}, \tag{4}$$

where each row uses a different root of unity ζ_p. We then note that

$$\begin{aligned}
\Phi_p'(\zeta_p) &= (\zeta_p - \zeta_p^2) \cdot (\zeta_p - \zeta_p^3) \cdots (\zeta_p - \zeta_p^{p-1}) \\
&= \zeta_p^{-2} \cdot (1 - \zeta_p) \cdot (1 - \zeta_p^2) \cdots (1 - \zeta_p^{p-2}) \cdot \frac{(1 - \zeta_p^{p-1})}{(1 - 1/\zeta_p)} \\
&= \frac{\zeta_p^{-2} p}{1 - 1/\zeta_p} = \frac{p}{\zeta_p^2 - \zeta_p}.
\end{aligned}$$

Thus the coefficients of the polynomial in (4) are given by $\zeta_p \cdot (\zeta_p^r - 1)/p$ for $r = 1, \ldots, p-1$. Where each row of our matrix is given by a different pth root ζ_p.

Thus to determine the infinity norm of CRT_p^{-1} we simply need to sum the absolute values of these coefficients, for the first row, since all other rows will be equal:

$$\begin{aligned}
C_p &= \sum_{r=1}^{p-1} |\zeta_p(\zeta_p^r - 1)/p| = \frac{1}{p} \sum_{r=1}^{p-1} \sqrt{2 - 2 \cdot \cos(2r\pi/p)} \\
&= \frac{1}{p} \sum_{r=1}^{p-1} 2 \cdot \sin(r\pi/p) = \frac{2 \cdot \sin(\pi/p)}{p \cdot (\cos(\pi/p) - 1)}
\end{aligned}$$

□

In practice this result means that $C_p \approx 4/\pi \approx 1.2732$ for all $p \geq 11$.

If m is odd then we see that, subject to a permutation of the rows, the matrix CRT_{2m} and CRT_m are identical up to a multiple of -1 for every second column. Thus we have

$$C_{2m} = C_m \text{ for odd values of } m.$$

We find that $C_r \leq 8.6$ for squarefree $r \leq 400$, which provides a relatively small upper bound on C_m for an infinite family of cyclotomic fields K. It appears that the size of C_m depends crucially on the number of prime factors of m. Thus it is an interesting open question to provide a tight upper bound on C_m. Indeed the growth in C_m seems to be closely related to the growth in the coefficients of the polynomial $\Phi_m(X)$, which also depends on the number of prime factors of m.

[7] This proof was provided to us by Robin Chapman .

C.3 Application of the above bounds

An immediate consequence of Theorem 7 is to provide an upper bound on the value δ_∞ for cyclotomic number fields. Let $\alpha \in \mathcal{O}_K$ then we have, by the standard inequalities between norms, that $\|\alpha\|_1 \leq N \cdot \|\alpha\|_\infty$. Thus we have, for $\alpha, \beta \in \mathcal{O}_K$,

$$\begin{aligned}\|\alpha \cdot \beta\|_\infty &\leq C_m \cdot \|\sigma(\alpha \cdot \beta)\|_\infty \leq C_m \cdot \|\sigma(\alpha)\|_\infty \cdot \|\sigma(\beta)\|_\infty \\ &\leq C_m \cdot \|\alpha\|_1 \cdot \|\beta\|_1 \\ &\leq C_m \cdot N^2 \cdot \|\alpha\|_\infty \cdot \|\beta\|_\infty,\end{aligned}$$

i.e. $\delta_\infty \leq C_m \cdot N^2$. When m is a power of two, since $C_m = 1$ we find the bound $\delta_\infty \leq \phi(m)^2$; however in this case it is known that $\delta_\infty = \phi(m)$, thus the above bound is not tight.

A more interesting application, for our purposes, is to bound the infinity norm in the polynomial embedding of the product of two elements which have been selected with a discrete Gaussian. To demonstrate this result we will first need to introduce the following standard tailbound:

Lemma 6. *Let $c \geq 1$ and $C = c \cdot \exp(\frac{1-c^2}{2}) < 1$ then for any integer $N \geq 1$ and real $r > 0$ we have*

$$\Pr_{\mathbf{x} \leftarrow D_{\mathbb{Z}^N, s}} \left[\|\mathbf{x}\|_2 \geq c \cdot s \cdot \sqrt{\frac{N}{2 \cdot \pi}} \right] \leq C^N.$$

Note that this implies that

$$\Pr_{\mathbf{x} \leftarrow D_{\mathbb{Z}^N, s}} \left[\|\mathbf{x}\|_2 \geq 2 \cdot r \cdot \sqrt{N} \right] \leq 2^{-N},$$

where $r = s/\sqrt{2 \cdot \pi}$. If we therefore select $\alpha, \beta \in D_{\mathbb{Z}^N, s}$, consider them as elements of $\mathcal{O}_K$, we then have, with overwhelming probability that $\|\alpha\|_2, \|\beta\|_2 \leq 2 \cdot r \cdot \sqrt{N}$. We then apply the standard inequality between the 2- and the 1-norm to deduce $\|\alpha\|_1, \|\beta\|_1 \leq 2 \cdot r \cdot N$. We then have that

$$\begin{aligned}\|\alpha \cdot \beta\|_\infty &\leq C_m \cdot \|\sigma(\alpha \cdot \beta)\|_\infty \leq C_m \cdot \|\sigma(\alpha)\|_\infty \cdot \|\sigma(\beta)\|_\infty \\ &\leq C_m \cdot \|\alpha\|_1 \cdot \|\beta\|_1 \\ &\leq 4 \cdot C_m \cdot r^2 \cdot N^2.\end{aligned}$$

D Security, Parameter Choice and Performance

In this Appendix we show that our concrete SHE scheme meets all the security requirements required by our MPC protocol, i.e. that it is an admissible cryptosystem. On the way we derive parameter settings, and finally we present some implementation results for the core operations.

Recall a cryptosystem is admissible if it meets the following requirements:

- It is IND-CPA secure.
- It has a KeyGen^* function with the required properties.
- It is (B_{plain}, B_{rand}, C)-correct, where $B_{plain} = N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{\mathsf{sec}/2+8}$, $B_{rand} = d \cdot \rho \cdot \mathsf{sec}^2 \cdot 2^{\mathsf{sec}/2+8}$, and where C, the set of functions we can evaluate on ciphertexts, contains all formulas evaluated in the protocol $\Pi_{\text{PREP}'}$ (including the identity function). Note that here we choose the values for B_{plain}, B_{rand} that correspond to the most efficient variant of the ZK proofs.

Recall in the expressions for B_{plain} and B_{rand} we have d is the dimension of the randomness space, i.e. $d = 3 \cdot N$, τ is a bound on the infinity norm of valid plaintexts, i.e. $p/2$; and ρ is a bound on the infinity norm of the randomness in validly generated ciphertexts, i.e. $\rho \approx 2 \cdot r \cdot \sqrt{N}$, by the tailbound of Lemma 6.

IND-CPA and KeyGen*'s properties: We first turn to discussing security. Since our scheme is identical (bar the distributed decryption functionality) to that of [7], security can be reduced to the hardness of the following problem.

Definition 2 (PLWE Assumption). *For all* $\mathsf{sec} \in \mathbb{N}$*, let* $f(X) = f_{\mathsf{sec}}(X) \in \mathbb{Z}[X]$ *be a polynomial of degree* $N = N(\mathsf{sec})$*, let* $q = q(\mathsf{sec}) \in \mathbb{Z}$ *be a prime integer, let* $R = \mathbb{Z}[X]/f(X)$ *and* $R = R/qR$*, and let* χ *denote a distribution over the ring* R*. The* polynomial LWE assumption $\mathsf{PLWE}_{f,q,\chi}$ *states that for any* $l = poly(\mathsf{sec})$ *it holds that*

$$\{(a_i, a_i \cdot s + e_i)\}_{i\in[l]} \approx \{(a_i, u_i)\}_{i\in[l]}$$

where s *is sampled from the distribution* χ*, and* a_i, u_i *are uniformly random in* R_q*. We require computational indistinguishability to hold given only* l *samples, for some* $l = poly(\mathsf{sec})$*.*

In particular our scheme is semantically secure if the $\mathsf{PLWE}_{\Phi_m(X),q_0,\mathsf{D}^N_\rho(s)}$-problem is hard. The hardness of the same problem also implies that the output from $\mathsf{KeyGen}()$ is computationally indistinguishable from that of $\mathsf{KeyGen}^*()$.

Thus our first task is to derive relationships between the parameters so as to ensure the first two properties of being admissible are satisfied, i.e. the PLWE problem is actually hard to solve. The basic parameters of our scheme are the degree of the associated number field $N = \phi(m)$, the standard deviation r of the used Gaussian distribution, and the modulus q. We first turn to estimating r; we do this by using the "standard" analysis of the underlying LWE problem.

We first ensure that r is chosen to avoid combinatorial style attacks. Consider the underlying LWE problem as being given by $\mathbf{s} \cdot A + \mathbf{e} = \mathbf{v}$, where $\mathbf{e}$ is the LWE error vector, and A is a random $N \times t$ matrix over $\mathbb{F}_q$. In [1] the authors present a combinatorial attack which breaks LWE in time $2^{O(\|\mathbf{e}\|^2_\infty)}$ with high probability. Since $\mathbf{e}$ is chosen by the discrete Gaussian with standard deviation r, if we pick r large enough then this attack should be prevented. Thus choosing r such that $r > 3.2$ will ensure that r is large enough to avoid combinatorial attacks, i.e. $s \geq 8$.

We now turn to the distinguishing problem, namely given $\mathbf{v}$ can we determine whether it arises from an LWE sample, or from a uniform sample. We determine a lower bound on N. The natural "attack" against the decision LWE problem is to first find a short vector $\mathbf{w}$ in the dual lattice $\Lambda_q(A^\mathsf{T})^*$ and then check whether $\mathbf{w} \cdot \mathbf{v}^\mathsf{T}$ is close to an integer. If it is then the input vector is an LWE sample, if not it is random. Thus to ensure security, following the argument in [23][Section 5.4.1], we require

$$r \geq \frac{1.5}{\|\mathbf{w}\|_2}.$$

Following the work of [14] we can estimate, for $t \gg N$, the size of the output of a lattice reduction algorithm operating on the lattice $\Lambda_q(A^\mathsf{T})^*$. In particular if the algorithm tries to find a vector with root Hermite factor δ (thus δ measures the difficulty in breaking the underlying SHE system, typically one may select $\delta \approx 1.005$, but see later for other choices) then we expect to find a vector $\mathbf{w}$ of size

$$\frac{1}{q}\min(q, \delta^t \cdot q^{N/t}).$$

Following the analysis of [23] the above quantity is minimized when we select $t = t' := \sqrt{N\log(q)/\log(\delta)}$. This leads us to the deduce the lower bound

$$r \geq 1.5 \cdot \max(1\ ,\ \delta^{-t'} \cdot q^{1-N/t'}).$$

Noise of a Clean Ciphertext: We now turn to determining the bound, in the infinity norm, of the value obtained in decrypting valid ciphertexts. Consider what happens when we decrypt a clean ciphertext, encrypted via $(\mathbf{c}_0, \mathbf{c}_1) = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}, \mathbf{r})$, with $\mathbf{r} = (\mathbf{u}, \mathbf{v}, \mathbf{w})$. This looks like a PLWE sample $(\mathbf{c}_1, \mathbf{c}_0)$ where the "noise" term, for a validly generated clean ciphertext, is given by

$$\begin{aligned}\mathbf{t} &= \mathbf{c}_0 - \mathbf{s} \cdot \mathbf{c}_1 \\ &= \mathbf{x} + p \cdot (\mathbf{e} \cdot \mathbf{v} + \mathbf{w} + \mathbf{s} \cdot \mathbf{u})\end{aligned}$$

By our estimates in Appendix C.3 we can bound the infinity norm of $\mathbf{t}$ by

$$\|\mathbf{t}\|_\infty \leq \frac{p}{2} + p \cdot \left(4 \cdot C_m \cdot r^2 \cdot N^2 + 2 \cdot \sqrt{N} \cdot r + 4 \cdot C_m \cdot r^2 \cdot N^2\right) =: Y.$$

(B_{plain}, B_{rand}, C)-correctness: Whilst IND-CPA is about security in relation to validly created ciphertexts, our distributed decryption functionality must be secure even when some ciphertexts are not completely valid. This was why we introduced the notion of (B_{plain}, B_{rand}, C)-correctness. We need to pick B_{plain} and B_{rand} so that $B_{plain} \geq N \cdot \tau \cdot \mathsf{sec}^2 \cdot 2^{\mathsf{sec}/2+8}$ and $B_{rand} \geq d \cdot \rho \cdot \mathsf{sec}^2 \cdot 2^{\mathsf{sec}/2+8}$. Since $B_{plain} \ll B_{rand}$ we estimate the noise term associated to such a "clean" ciphertext will be bounded by $Y' = (B_{rand}/\rho)^2 \cdot Y = 9 \cdot N^2 \cdot \mathrm{sec}^4 \cdot 2^{\mathsf{sec}+16} \cdot Y$. In our MPC protocol we only need to be able to evaluate functions of the form

$$(x_1 + \cdots + x_n) \cdot (y_1 + \cdots + y_n) + (z_1 + \cdots + z_n).$$

We can, via the results in Appendix C.3, crudely estimate the size of B, from Section 6, needed to ensure valid decryption. Our crude (over-) estimate therefore comes out as

$$\begin{aligned}B &\leq \delta_\infty \cdot (n \cdot Y') \cdot (n \cdot Y') + (n \cdot Y') \\ &\leq C_m \cdot N^2 \cdot n^2 \cdot Y'^2 + n \cdot Y' \\ &\leq C_m \cdot N^2 \cdot n^2 \cdot c_{\mathsf{sec}}^2 \cdot Y^2 + n \cdot c_{\mathsf{sec}} \cdot Y =: Z\end{aligned}$$

where $c_{\mathsf{sec}} = 9 \cdot N^2 \cdot \mathsf{sec}^4 \cdot 2^{\mathsf{sec}+8}$. We take Z as the bound, which we then need to scale by $1 + 2^{\mathsf{sec}}$ to ensure we have sufficient space to enable the distributed decryption algorithm. Hence, the value of q needs to be selected so that $Z \cdot (1 + 2^{\mathsf{sec}}) < q/2$.

So in summary we need to choose parameters such that

$$\begin{aligned}&q > 2 \cdot Z \cdot (1 + 2^{\mathsf{sec}}), \\ &r > \max\left\{3.2, 1.5 \cdot \delta^{-t'} \cdot q^{1-N/t'}\right\},\end{aligned}$$

where sec is the statistical security parameter, δ is a measure of how hard it is to break the underlying SHE scheme, and $t' = \sqrt{N \log(q)/\log(\delta)}$. This leads to a degree of circularity in the dependency of the parameters, but valid parameter sets can be found by a simple search technique.

Specific Parameter Sets: To determine parameters for fixed values of $(\mathbb{F}_{p^k})^s$ and n we proceed as follows. There are two interesting cases; one where p is fixed (i.e. $p = 2$) and one where we only care that p is larger than some bound (i.e. $p > 2^{32}$, or $p > 2^{64}$). The latter case of $p > 2^{64}$ is more interesting as such size numbers can be utilized more readily in applications since we can simulate integer arithmetic without overflow with such numbers. In addition using such a value of p means we do not need to repeat our ZKPoKs, or replicate the MACs so as to get a cheating probability of less than 2^{-40}.

Our method in all cases is to first fix p, n, sec and δ, we then search using the above inequalities for (rough) values of q and N which satisfy the inequalities above. We then search for exact values of p and N which satisfy our functional requirements on p (i.e. fixed or greater than some bound) plus N larger than the bound above, such that N is the degree of $F(X) = \Phi_m(X)$ and $F(X)$ splits into at least s factors of degree divisible by k over $\mathbb{F}_p$.

Given this precise value for N, we then return to the above inequalities to find exact values of q and r. In all our examples below we pick $n = 3$, $\mathsf{sec} = 40$, and $\delta = 1.0052$.

Example 1. We first look at $p > 2^{32}$. Our first (approximate) search reveals we need $N > 14300$, $q \approx 2^{430}$ and $r = 3.2$ (assuming $C_m \leq 2$). We then try to find an optimal value of N; this is done by taking increasing primes $p > 2^{32}$ and factoring $p - 1$. The factors of $p - 1$ correspond to values of m such that $\Phi_m(X)$ factors into $\phi(m)$ factors modulo p. So we want to find a p such that $p - 1$ is divisible by an m, so that $N = \phi(m) > 14300$. A quick search reveals candidates of

$$(p, N, m) = (2^{32} + 32043,\ 14656,\ 14657).$$

Picking m in this way will maximise the value of $s = n$, and hence allow us to perform more operations in parallel. In addition since m is prime we know, by Lemma 5, that $C_m \approx 1.2732$, thus justifying our assumption in deriving the bounds of $C_m \leq 2$.

Selecting m to be the prime 14657 in addition allows us to evaluate $s = p-1 = 14656$ runs of the triple production algorithm in parallel. The message expansion factor, given we require $N \cdot \log_2(q)$ bits to represent N elements in $\mathbb{F}_p$ is given by

$$\frac{N \cdot \log_2(q)}{N \cdot \log_2(p)} = \frac{\log_2(q)}{\log_2(p)} = \frac{430}{32} \approx 13.437.$$

Example 2. Performing the same analysis for a $p > 2^{64}$, our first naive search of parameters reveal we need an $n \approx 16700$ and $q \approx 2^{500}$. We then search for specific parameters and find $p = 2^{64} + 4867$ is pretty near to optimum, which results in a prime value of m of 16729. We find the expansion factor is given by

$$\frac{\log_2(q)}{\log_2(p)} = \frac{500}{64} \approx 7.81.$$

Example 3. We now look at the case $p = 2$ and $k = 8$, i.e. we are looking for parameters which would allow us to compute AES circuits in parallel; or more generally circuits over $\mathbb{F}_{2^8}$. Our first approximate search reveals that we need $N > 12300$, $q \approx 2^{370}$ and $r = 3.2$. So we now need to determine a value m such that

$$N = \phi(m) > 12100 \text{ and } 2^d \equiv 1 \pmod{m} \text{ and } d \equiv 0 \pmod{8}.$$

A quick search reveals candidates of

$$(m, N) = (17425, 12800)$$

since $\Phi_{17425}(X)$ factors into $s = 320$ factors of degree $d = 40$ modulo 2. Thus using this value of m we are able to work with $s = 320$ elements of $\mathbb{F}_{2^8}$ in parallel. The message expansion factor, given we require $N \cdot \log_2(q)$ bits to represent 320 elements in $\mathbb{F}_{2^8}$ is given by

$$\frac{N \cdot \log_2(q)}{8 \cdot s} = \frac{d \cdot 370}{8} = 1850.0$$

For this value of m we find $C_{17425} \approx 9.414$.

We present the following run-times we have achieved. We time the operations for encrypting and decrypting clean ciphertexts, the time to homomorphically compute $(c_x \boxtimes c_y) \boxplus c_z$, plus the time to decrypt the said result. The times are given in seconds, and in brackets we present the amortized time per finite field element. All timings were performed on an Intel Core-2 6420 running at 2.13 GHz.

Example	Enc Time (s)	Dec (Clean) Time (s)	$(c_x \boxtimes c_y) \boxplus c_z$ Time (s)	$\mathsf{Dec}_{\mathsf{sk}}((c_x \boxtimes c_y) \boxplus c_z)$ Time (s)
1	0.72 {0.00005}	0.35 {0.00002}	1.43 {0.0001}	0.72 {0.00005}
2	3.13 {0.00019}	1.54 {0.00009}	6.27 {0.0004}	3.15 {0.00018}
3	1.26 {0.00394}	0.60 {0.00188}	2.46 {0.0077}	1.23 {0.00384}

Estimating Equivalent Symmetric Security Level: The above examples were computed using the root Hermite factor of $\delta = 1.005$. Mapping this "hardness" parameter for the underlying lattice problem to a specific symmetric security level (i.e. 80-bit security, or 128-bit security) is a bit of a "black art" at present.

In [9] the authors derive an estimate for the block size needed to obtain a given root Hermite factor, assuming an efficient BKZ lattice reduction algorithm is used. They then provide estimates as to the run time needed for a specific enumeration using this block size. As an example of their analysis they estimate that a block size of 286 is needed to obtain a root Hermite factor of $\delta = 1.005$. Then they estimate that the run time needed to perform the enumeration in a projected lattice of such dimension (the key sub-procedure of the BKZ algorithm) takes time roughly between 2^{80} and 2^{175} operations. Thus a value of $\delta = 1.005$ can be considered secure; however their estimates are not precise enough to produce parameters associated with a given symmetric security level.

In [20] the authors take a different approach and simply extrapolate run times for the NTL implementation of BKZ. By looking at various LWE instances, they derive the following equation linking the expected run-time of a distinguishing attack and the root Hermite factor

$$\log_2 T = \frac{1.8}{\log_2 \delta} - 110.$$

The problem with this approach is that NTL's implementation of BKZ is very old, and hence is not state-of-the-art; on the other hand we are able to derive a direct linkage between δ and $\log_2 T$. Using this equation we find the following equivalences:

$\log_2 T$	80	100	128	196	256
δ	1.0066	1.0059	1.0052	1.0041	1.0034

Using these estimates for δ we re-run the above analysis to find approximate values for N and q in our three example applications; again assuming $n = 3$ and $\mathsf{sec} = 40$.

	$\mathbb{F}_p : p > 2^{32}$		$\mathbb{F}_p : p > 2^{64}$		$\mathbb{F}_{2^8}$	
	$N >$	$\log_2 q \approx$	$N >$	$\log_2 q \approx$	$N >$	$\log_2 q \approx$
$\delta = 1.0066$	11300	430	12900	490	9500	360
$\delta = 1.0059$	12600	430	14700	500	10900	370
$\delta = 1.0052$	14300	430	16700	500	12300	370
$\delta = 1.0041$	18600	440	21100	500	15600	370
$\delta = 1.0034$	22400	440	25500	500	18800	370

As can be seen the security parameter has only marginal impact on $\log_2 q$, and results in a doubling of the size of N as we increase from a security level of 80 bits to 256 bits. As a comparison if we, for security level 128 bits, i.e. $\delta = 1.0052$, increase the value of sec from 40 to 80 we find the following parameter sizes:

	$\mathbb{F}_p : p > 2^{32}$		$\mathbb{F}_p : p > 2^{64}$		$\mathbb{F}_{2^8}$	
	$N >$	$\log_2 q \approx$	$N >$	$\log_2 q \approx$	$N >$	$\log_2 q \approx$
$\delta = 1.0052$	18700	560	21000	630	16700	500

E Functionalities

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

Random Sample: When receiving $(rand)$ from all parties, it samples a uniform $r \leftarrow \{0,1\}^u$ and outputs $(rand, r)$ to all parties.

Random modulo p: When receiving $(rand, p)$ from all parties, it samples a uniform value $e \leftarrow \mathbb{F}_{p^k}$ and outputs $(rand, e)$ to all parties.

Fig. 14. The ideal functionality for coin-flipping.

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

Initialize: On input $(init, p^k)$ from all parties, the functionality activates and stores the modulus p.

Input: On input $(input, P_i, varid, x)$ from P_i and $(input, P_i, varid, ?)$ from all other parties, with $varid$ a fresh identifier, the functionality stores $(varid, x)$.

Add: On command $(add, varid_1, varid_2, varid_3)$ from all parties (if $varid_1, varid_2$ are present in memory and $varid_3$ is not), the functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x + y \mod p)$.

Multiply: On input $(multiply, varid_1, varid_2, varid_3)$ from all parties (if $varid_1, varid_2$ are present in memory and $varid_3$ is not), the functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x \cdot y \mod p)$.

Output: On input $(output, varid)$ from all honest parties (if $varid$ is present in memory), the functionality retrieves $(varid, x)$ and outputs it to the environment. If the environment inputs OK then x is output to all players. Otherwise $\perp$ is output to all players.

Fig. 15. The ideal functionality for arithmetic MPC.

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

Usage: We first describe two macros, one to produce $[\![\mathbf{v}]\!]$ representations and one to produce $\langle\mathbf{v}\rangle$ representations. We denote by A the set of players controlled by the adversary.

$\mathsf{Bracket}(\mathbf{v}_1,\ldots,\mathbf{v}_n,\Delta_1,\ldots,\Delta_n,\beta_1,\ldots,\beta_n)$, where $\mathbf{v}_1,\ldots,\mathbf{v}_n,\Delta_1,\ldots,\Delta_n \in (\mathbb{F}_{p^k})^s$, $\beta_1,\ldots,\beta_n \in \mathbb{F}_{p^k}$

1. Let $\mathbf{v} = \sum_{i=1}^n \mathbf{v}_i$
2. For $i = 1,\ldots,n$
   (a) The functionality computes the MAC $\gamma(\mathbf{v})_i \leftarrow \mathbf{v}\cdot\beta_i$ and sets $\gamma_i \leftarrow \gamma(\mathbf{v})_i + \Delta_i$
   (b) For every corrupt player P_j, $j \in A$ the environment specifies a share γ_i^j
   (c) The functionality sets each share γ_i^j, $j \notin A$, uniformly such that $\sum_{j=1}^n \gamma_i^j = \gamma_i$
3. The functionality sends $(\mathbf{v}_i, (\beta_i, \gamma_1^i,\ldots,\gamma_n^i))$ to each honest player P_i (dishonest players already have the respective data).

$\mathsf{Angle}(\mathbf{v}_1,\ldots,\mathbf{v}_n,\Delta,\alpha)$, where $\mathbf{v}_1,\ldots,\mathbf{v}_n,\Delta \in (\mathbb{F}_{p^k})^s$, $\alpha \in \mathbb{F}_{p^k}$

1. Let $\mathbf{v} = \sum_{i=1}^n \mathbf{v}_i$
2. The functionality computes the MAC $\gamma(\mathbf{v}) \leftarrow \alpha\cdot\mathbf{v}$ and sets $\gamma \leftarrow \gamma(\mathbf{v}) + \Delta$
3. For every corrupt player P_i, $i \in A$ the environment specifies a share γ_i
4. The functionality sets each share γ_i $i \notin A$ uniformly such that $\sum_{i=1}^n \gamma_i = \gamma$
5. The functionality sends $(0, \mathbf{v}_i, \gamma_i)$ to each honest player P_i (dishonest players already have the respective data).

Initialize: On input $(init, p, k, s)$ from all players, the functionality stores the prime p and the integers k, s. It then waits for the environment to call either "stop" or "OK". In the first case the functionality sends "fail" to all honest players and stops. In the second case it does the following:

1. For each corrupt player P_i, $i \in A$, the environment specifies a share α_i
2. The functionality sets each share α_i, $i \notin A$ uniformly
3. For each corrupt player P_i, $i \in A$, the environment specifies a key β_i
4. The functionality sets each key β_i $i \notin A$ uniformly
5. The environment specifies $\Delta_1,\ldots,\Delta_n \in (\mathbb{F}_{p^k})^s$
6. It runs the macro $\mathsf{Bracket}(\mathsf{Diag}(\alpha_1),\ldots,\mathsf{Diag}(\alpha_n),\Delta_1,\ldots,\Delta_n,\beta_1,\ldots,\beta_n)$.

Pair: On input $(pair)$ from all players, the functionality waits for the environment to call either "stop" or "OK". In the first case the functionality sends "fail" to all honest players and stops. In the second case it does the following:

1. For each corrupt player P_i, $i \in A$, the environment specifies a share $\mathbf{r}_i$
2. The functionality sets each share $\mathbf{r}_i$, $i \notin A$ uniformly
3. The environment specifies $\Delta, \Delta_1,\ldots,\Delta_n \in (\mathbb{F}_{p^k})^s$
4. It runs the macros $\mathsf{Bracket}(\mathbf{r}_1,\ldots,\mathbf{r}_n,\Delta_1,\ldots,\Delta_n,\beta_1,\ldots,\beta_n)$ and $\mathsf{Angle}(\mathbf{r}_1,\ldots,\mathbf{r}_n,\Delta,\alpha)$.

Triple: On input $(triple)$ from all players, the functionality waits for the environment to call either "stop" or "OK". In the first case the functionality sends "fail" to all honest players and stops. In the second case it does the following

1. For each corrupt player P_i, $i \in A$, the environment specifies shares $\mathbf{a}_i, \mathbf{b}_i$
2. The functionality sets each share $\mathbf{a}_i, \mathbf{b}_i$, $i \notin A$ uniformly. Let $\mathbf{a} := \sum_{i=1}^n \mathbf{a}_i$, $\mathbf{b} := \sum_{i=1}^n \mathbf{b}_i$
3. The environment specifies $\Delta_{\mathbf{a}}, \Delta_{\mathbf{b}}, \delta \in (\mathbb{F}_{p^k})^s$
4. It sets $\mathbf{c} \leftarrow \mathbf{a}\cdot\mathbf{b} + \delta$
5. For each corrupt player P_i, $i \in A$, the environment specifies shares $\mathbf{c}_i$
6. The functionality sets each share $\mathbf{c}_i$, $i \notin A$ uniformly with the constrain $\sum_{i=1}^n \mathbf{c}_i = \mathbf{c}$
7. The environment specifies $\Delta_{\mathbf{c}} \in (\mathbb{F}_{p^k})^s$
8. It runs the macros $\mathsf{Angle}(\mathbf{a}_1,\ldots,\mathbf{a}_n,\Delta_{\mathbf{a}},\alpha)$, $\mathsf{Angle}(\mathbf{b}_1,\ldots,\mathbf{b}_n,\Delta_{\mathbf{b}},\alpha)$, $\mathsf{Angle}(\mathbf{c}_1,\ldots,\mathbf{c}_n,\Delta_{\mathbf{c}},\alpha)$.

Fig. 16. The ideal functionality for making the global key $[\![\alpha]\!]$, pairs $[\![\mathbf{r}]\!], \langle\mathbf{r}\rangle$ and triples $\langle\mathbf{a}\rangle, \langle\mathbf{b}\rangle, \langle\mathbf{c}\rangle$