Multiparty Computation from Somewhat Homomorphic Encryption

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

[1] Aarhus
[2] Bristol

Abstract. We propose a general multiparty computation protocol secure against a dishonest majority, for computing securely arithmetic circuits over a finite field $\mathbb{F}_{p^k}$. As in several earlier works, 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. Our preprocessing is based on a somewhat homomorphic cryptosystem. We extend a scheme by Brakersky et al., allowing us to perform distributed decryption and to handle many values in parallel. Our preprocessing phase improves significantly over earlier work both asymptotically and in practice. The online phase may use an existing protocol by Bendlin et al., based on unconditionally secure MACs, but we also propose a new online phase that scales better with n, the number of players. The total amount of data the players need to store from the preprocessing is linear in n rather than quadratic as in earlier work. Furthermore, the cost of a secure multiplication in our online phase is $O(n)$ multiplications in $\mathbb{F}_{p^k}$ plus $O(n^2)$ additions, rather than $O(n^2)$ multiplications as in earlier work.

1 Introduction

A central problem in theoretical cryptography is that of secure multiparty computation (MPC). In this problem n parties, holding private inputs $x_1, ..., x_n$, 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 [8] it was shown how to construct UC-secure MPC protocols that are composable and handle the case where all but one of the parties are actively corrupted. In this case one needs public-key machinery that is typically expensive, so designing efficient solutions becomes more difficult. Recently, however, a number of improvements have been proposed making such protocols eminently practical. The standard, modern, approach works as follows. The function to be computed is expressed as an arithmetic circuit over a given finite field. At the beginning of a protocol the parties engage in an offline, or preprocessing, phase which is (essentially) independent of the function being computed, where raw material for later use is produced using public-key techniques. Then in an online phase the parties evaluate the function; this phase can be very efficient using only cheap information theoretic primitives, since the players use data produced in the preprocessing. The current state of the art in this respect are the protocols in [10, 3, 7].

Recently, another approach has become possible with the advent of Fully Homomorphic Encryption (FHE) [12]. In this approach all parties first encrypt their input under the FHE scheme; these encryptions are broadcast to all players. Then the arithmetic circuit representing the function f is evaluated homomorphically by each player locally. Finally a distributed decryption is performed on the common final ciphertexts held by each player. The advantage of the FHE-based approach is that the only needed interaction is when parties commit to their inputs via the initial encryptions and when they perform distributed decryption. However, 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), unless one assumes circular security and one implements an expensive bootstrapping operation. The main cost, much like the MPC based approach above, 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.

In this paper we combine the two approaches and obtain a solution with some attractive properties from both worlds. More concretely, we present two improvements of previous work on secure arithmetic computa-

tion. First, we improve performance of the preprocessing phase by reducing both interaction and computation using an SHE scheme, an adaptation of the LWE-based scheme of [5]. Whereas [10, 3] use $O(n^2)$ public-key operations per secure multiplication, we use $O(n^2/s)$ operations, where s is a number that grows with the security parameter of the SHE scheme (we have $s \approx 15000$ in our concrete instantiation). We stress that our adapted scheme is exactly as efficient as the basic version of [5] that does not allow this optimization, so the improvement is indeed "genuine". Second, we improve the performance of the online phase of [3] which does not scale as well as one might hope with n, the number of players. If m_f is the number of multiplications we need to evaluate f in the online phase, then the amount of data players need to store from the preprocessing is $\Theta(m_f \cdot n^2)$. We propose an new online phase in which only $O(m_f \cdot n)$ values have to be stored. One must expect that m_f can be quite large for a practical application, so the difference between $m_f \cdot n$ and $m_f \cdot n^2$ can be quite significant already for moderate values of n. Furthermore, the complexity of the online phase is also improved: the cost of secure multiplication is $O(n)$ multiplications in $\mathbb{F}_{p^k}$ plus $O(n^2)$ additions, rather than $O(n^2)$ multiplications as in earlier work.

To understand our approach for the offline phase, one needs first a high level view of the offline phase of [3] (and other similar protocols). The offline phase's role is to produce a set of "multiplication triples", namely a secret sharing of values (a, b, c) in some finite field such that $c = a \cdot b$. Players are committed to their shares using information theoretically secure authentication tags. During the online phase one triple is consumed for each multiplication gate, via the classic circuit randomization technique of Beaver [2]. Traditionally (and also in [3]) the triples are produced by relatively complex protocols and zero-knowledge proofs. We dispense with the most expensive protocol from [3], the protocol for proof of correct multiplication. Instead we produce the multiplicative properties in both the triples and the tags via the SHE scheme. We require a distributed decryption procedure, and we design such a procedure tailored for our cryptosystem. While this procedure is quite straightforward it is only robust against passive attacks. Nevertheless, we show – perhaps somewhat surprisingly – that this is sufficient for the overall protocol to be actively secure.

Our approach offers several advantages over the pure FHE technique: First, by applying the SHE SIMD approach of [20] to the cryptosystem from [5] we obtain a scheme where each ciphertext contains several values from the field. Then, we can produce multiple triples for the price of one SHE multiplication of ciphertexts, independently of the arithmetic circuit for f. In comparison using FHE SIMD operations in a purely FHE-based setting would only be applicable for a certain type of function f; i.e. one which is amenable for SIMD computation. Second, we use SHE only in the preprocessing phase; once the function to be evaluated is known we switch to the information theoretic and much more efficient online MPC approach. Third, we only require the underlying SHE scheme to be able to evaluate circuits of multiplicative depth one, and of relatively small additive complexity. Therefore we can choose parameters for the cryptosystem in a less aggressive way, and we do not need circular security or bootstrapping.

These facts lead to much better performance than one normally expects from protocols based on SHE, and also better than previous work for secure computation of arithmetic circuits. Our online phase is similar to those in [3] and [7], but more efficient (as explained above). Whilst [7] deals with a slightly different situation of boolean circuits we note that [7] reports on evaluating a 32000 gate circuit in under two seconds; we can expect our approach to obtain a similar performance for general arithmetic circuits. We now turn to the offline phase, where we use SHE-based techniques. To demonstrate that using SHE in this application does indeed improve the performance compared to previous techniques we present sample running times in Appendix C. Based on our timings and a count of operations, we estimate (for the case of two parties) that it will take less than 8 ms to prepare one multiplication in $\mathbb{F}_p$ for a 64-bit p, security level corresponding roughly to 1024 bit RSA and an error probability of 2^{-40} for the zero-knowledge proofs. This is 2-3 orders of magnitude faster than preliminary estimates for the most efficient instantiation of [3].

In recent independent work [18], Meyers at al. also use an FHE scheme for multiparty computation. This is essentially the pure FHE approach mentioned above with a threshold decryption protocol tailored to the specific

FHE scheme. That construction, however, has a number of drawbacks: it is only for the honest majority setting, it does not have an efficient online phase, and finally it is based on a much less practical FHE scheme.

We end this 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 $\|\mathbf{x}\|_\infty := \max_{1 \le i \le n} |x_i|$, $\|\mathbf{x}\|_1 := \sum_{1 \le i \le n} |x_i|$ 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 induced from the random coins of A and its algorithm. Finally $x := y$ means "x is defined to be y".

2 The Abstract Somewhat Homomorphic Encryption Scheme

In this section we specify the properties we need for our cryptosystem; any scheme with these properties will be sufficient. A concrete instantiation is found in Section 5.

We first define the plaintext space M. This will be given by a direct produce of finite fields $(\mathbb{F}_{p^k})^s$ of characteristic p. Componentwise addition and multiplication of elements in M will be denoted by $+$ and $\cdot$.

However, we will not be working directly with elements in $\mathbb{F}_{p^k}$; instead 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 $\mathsf{decode}(\mathsf{encode}(\mathbf{m})) = \mathbf{m}$, but we do not require that $\mathsf{encode}(\mathsf{decode}(\mathbf{x})) = \mathbf{x}$ for all $\mathbf{x} \in \mathbb{Z}^N$ (indeed it is clear that this cannot hold). We will require that the decode operation is compatible with the characteristic of the field, i.e. for any $\mathbf{x} \in \mathbb{Z}^N$ we have $\mathsf{decode}(\mathbf{x}) = \mathsf{decode}(\mathbf{x} \pmod p)$. And finally that the encoding function does not produce large vectors. More precisely we have, for all $\mathbf{m} \in (\mathbb{F}_{p^k})^s$, that $\|\mathsf{encode}(\mathbf{m})\|_\infty \le \tau$ where $\tau = cp$ for some small constant c (say $1/2$).

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 (in our abstract scheme) as to what the representation of multiplication is for the ring R. 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.

Our basic abstract scheme is defined by the following tuple of algorithms

$$(\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{Enc}, \mathsf{Dec}),$$

which are defined as follows, and are parametrized by a security parameter κ. In addition, our protocols make use of an "information theoretic" security parameter sec, controlling the error probability for zero-knowledge proofs, MACs, etc. We will assume that p^k is $\Omega(2^{\mathsf{sec}})$ in our main presentation. Our protocols can also work with smaller values of p^k but will be less efficient, in that multiple MACs will need to be constructed and checked in the main protocol. The cost of the protocols for small values of p^k will grow by a constant equal to $\mathsf{sec}/\log_2(p^k)$.

$\underline{\mathsf{ParamGen}(1^\kappa, M)}$: 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^d_ρ, 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. D_ρ^d is used by the encryption algorithm to select the random coins needed during encryption. The algorithm 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 circuits over $\mathbb{F}_{p^k}$, 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 $\mathbb{F}_{p^k}[X_1, X_2, \ldots]$. 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 and then applying $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}, \mathbf{r})$, i.e.

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

$\mathsf{Dec}_{\mathsf{sk}}(c)$: This takes as input the secret key and a ciphertext c and 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 tries to decrypt 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. More formally, the scheme is said to be (B_{plain}, B_{rand}, C)-correct if

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

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 encodings and randomness that are bounded by B_{plain} and B_{rand}.

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 1. 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[1]. 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. 1. The Ideal Functionality for Distributed Key Generation

We note that it is possible to make do with a weaker set-up assumption, such as a common reference string (CRS), namely by 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.

The functionality $\mathcal{F}_{\text{KEYGENDEC}}$ in Figure 2, 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. 2. The Ideal Functionality for Distributed Key Generation and Decryption

[1] $\mathcal{F}_{\text{KEYGEN}}$ allows the adversary to decide which shares he wants to get of the secret key. While it may seem more natural to let the functionality do the sharing on its own, we need to define the functionality this way so that it can be implemented. A simulator trying to simulate a given protocol will have to make sure that the view of the protocol it generates for the adversary is consistent with what the functionality says to the honest players. This is not possible if the functionality decides on all shares on its own. Informally, what we model here is that we don't care which shares the adversary gets, as long as the secret is safe.

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.

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 $B_{plain} = \tau 2^{2\mathsf{sec}+\log \mathsf{sec}}$, $B_{rand} = \rho 2^{2\mathsf{sec}+\log \mathsf{sec}}$; and 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.

3 Zero-Knowledge Proof of Plaintext Knowledge

In this section we make use of a protocol that takes as input sec ciphertexts $c_i,\ i = 1, \ldots, \mathsf{sec}$ generated by one of the players in our protocol, who will act as the prover. If the prover is honest then $c_i = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_i, \mathbf{r}_i)$, where $\mathbf{x}_i$ has been obtained from the encode function, i.e. $\|\mathbf{x}_i\|_\infty \leq \tau$, and $\mathbf{r}_i$ has been generated from D_ρ^d (so we may assume that $\|\mathbf{r}_i\|_\infty \leq \rho$). We present a zero-knowledge proof of plaintext knowledge (ZKPoPK), following the protocol of [3], for the following relation:

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

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

In our preprocessing protocol below, players will be required to give such a ZKPoPK for all ciphertexts they provide as input. 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 protocol is described in Appendix B.1, Figure 7 and its computational complexity per ciphertext is essentially the cost of a constant number of encryptions. In the same Appendix B.1 we prove, using standard arguments, the following theorem

Theorem 1. *The protocol Π_{ZKPoPK} (Appendix B.1, Figure 7) is a zero-knowledge proof of knowledge for the relation R_{PoPK}.*

4 MPC Protocol From The Abstract Scheme

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 13 in Appendix D. Our protocol is structured in a preprocessing (or offline) phase and an online phase. For the latter we will use the construction from [3] while our preprocessing phase is a new and more efficient solution than that of [3]. After presenting the preprocessing phase, we present a new online phase that improves on [3] when the number of players is large.

4.1 Online Phase

Before describing our construction we give a brief overview of the online phase toward which our preprocessing is aimed. A central part is a certain representation of values in $\mathbb{F}_{p^k}$, written $[\cdot]$. First, each player P_i has $n-1$ random values α_j^i for $j = 1, \ldots, n$ with $i \neq j$. Then, for the representation of each data item (say $[x]$) each player P_i holds an additive share $x_i \in \mathbb{F}_{p^k}$ (such that $x = \sum_i x_i$) and a random value $\beta_{x,j}^i$. The pair $K_{x,j}^i = (\alpha_j^i, \beta_{x,j}^i)$ will be used to MAC the value x_j via $\mathsf{MAC}^i(x_j) := \alpha_j^i \cdot x_j + \beta_{x,j}^i$.

When the players hold this information on each data item, they are able to do linear computations obtaining new representations, where the additive sharing and the MACs will ensure that the values remain secret and are computed correctly. This means in particular that the players having representations $[x], [y]$ can compute $[x] + [y] = [x + y]$.

To compute a multiplication $[x \cdot y]$ we need representations, $[a], [b], [c]$, of multiplicative triples [2], where $a, b, c \in \mathbb{F}_{p^k}$ and $c = a \cdot b$. These triples are exactly what the preprocessing phase should produce. For details on computing with this representation and the online phase, see Section 3 in [3].

Jumping a bit ahead, a point should be made about the preprocessing: at the end it is not actually guaranteed that the honest players end up holding correct MACs on their values. Errors might be introduced by the adversary during the (distributed) decryption. However, this does not cause a problem for the online phase. If some honest player shows an incorrect MAC, the protocol simply aborts. This is not a restriction, since with dishonest majority we cannot guarantee termination anyway, the adversary can always cause the protocol to abort. This weaker guarantee of the preprocessing does not change the concrete protocol needed for the online phase, although to prove security, a change of the simulator in the proof is needed. The full online protocol Π_{AMPC} and the proof of the following Theorem is provided in Appendix B.2.

Theorem 2. *In the $\mathcal{F}_{\text{TRIP}}$-hybrid model, the protocol Π_{AMPC} implements $\mathcal{F}_{\text{AMPC}}$ with statistical security against any static[2], active adversary corrupting up to $n-1$ parties.*

4.2 The Preprocessing Phase

In this section we show how to produce the representations of the multiplicative triples $[a], [b], [c]$ needed in the online phase. That is, for a (and similarly for b and c) we need to compute for player i his share of a, MACs on this share for each of the other players and keys for the MACs on each of the other players' shares of a, that is the following data: $a_i, \left\{\mathsf{MAC}^j(a_i)\right\}_{j=1,j\neq i}^n, \left\{K_{a,j}^i\right\}_{j=1,j\neq i}^n$.

More precisely, we construct the protocol Π_{TRIP} which securely implements the functionality $\mathcal{F}_{\text{TRIP}}$ (specified in Figure 14 in Appendix D) in the presence of the key generation and decryption functionality $\mathcal{F}_{\text{KeyGenDec}}$ (Figure 2) described in Section 2 and the functionality $\mathcal{F}_{\text{Rand}}$ (Figure 12 in Appendix D) that creates random challenges. We present our construction using our abstract cryptosystem above with $s = 1$. Recall that s is the number of plaintext values that can be contained in a single ciphertext. Processing $s > 1$ elements at once follows immediately from what we describe below.

We use a subprotocol $\Pi_{\text{TRIP}'}$ in Figure 4 for creating $[a], [b], [c]$, where, however, it is not guaranteed that $a \cdot b = c$. In the main protocol Π_{TRIP} in Figure 3 we ensure that the multiplicative relation holds by constructing two triples and "sacrificing" one to check the other.

Theorem 3 below is proved in Appendix B.3.

Theorem 3. *The protocol Π_{TRIP} implements $\mathcal{F}_{\text{TRIP}}$ 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.*

[2] Π_{AMPC} can actually be shown to adaptively secure, but our implementation of $\mathcal{F}_{\text{TRIP}}$ will only be statically secure.

Protocol Π_{TRIP}

Initialize:

1. The players call "start" on $\mathcal{F}_{\text{KeyGenDec}}$ to obtain the public key pk.
2. Each player P_i generates α_j^i for $j \neq i$. (Recall that α_j^i is the first part in the key used to MAC a value held by P_j. This part of the key is fixed for all values to be MAC'ed, and thus α_j^i can be set up once in the beginning of the protocol.)
3. Each player P_i computes and broadcasts $e_{\alpha_j^i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\alpha_j^i)$.
4. Each player P_i invokes Π_{ZKPoPK} acting as prover on input $(e_{\alpha_j^i}, \ldots, e_{\alpha_j^i})$, where $e_{\alpha_j^i}$, is repeated sec times since this 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.)

Triple:

1. The players invoke $\Pi_{\text{TRIP}'}$ twice to obtain $[a], [b], [c]$ and $[f], [g], [h]$.
2. The players check that indeed $a \cdot b = c$ by "sacrificing" the other triple (f, g, h):
   - The players invoke $\mathcal{F}_{\text{Rand}}$ to get a random challenge e.
   - The players evaluate homomorphically the functions $\widehat{f_1}$ and $\widehat{f_2}$, where $f_1 := e \cdot [a] - [f]$ and $f_2 := [b] - [g]$, to obtain α and β.
   - The players open α (resp. β) to reveal ϵ (resp. δ).
   - The players now evaluate homomorphically the function $\widehat{f_3}$, where $f_3 := e \cdot [c] - [h] - \delta \cdot [f] - \epsilon \cdot [g] - \epsilon \cdot \delta$, and open the result.
   - If the result not zero then the players abort, otherwise the players output $[a], [b], [c]$.

Fig. 3. The protocol for constructing a reliable triple.

Protocol $\Pi_{\text{TRIP}'}$

Usage. The Triple-step below 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 $\Pi_{\text{TRIP}'}$ 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 $\Pi_{\text{TRIP}'}$ on our packed plaintext space of s elements of $\mathbb{F}_{p^k}$. Thus, we generate $s \cdot \mathsf{sec}$ elements in one go and then buffer the generated triples, outputting the next unused one on demand.

Triple:

1. Player P_i generates a_i, b_i, and r_i at random from $\mathbb{F}_{p^k}$.
2. Player P_i generates $K_{a,j}^i, K_{b,j}^i$ and $K_{c,j}^i$ for $j \neq i$, where $K_{a,j}^i = (\alpha_j^i, \beta_{a,j}^i)$, $K_{b,j}^i = (\alpha_j^i, \beta_{b,j}^i)$, $K_{r,j}^i = (\alpha_j^i, \beta_{r,j}^i)$.
3. Player P_i computes and broadcasts the ciphertexts

$$e_{a,i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(a_i), \;\; e_{b,i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(b_i), \;\; e_{r,i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(r_i),$$

$$e_{\beta_{a,j}^i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\beta_{a,j}^i), \;\; e_{\beta_{b,j}^i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\beta_{b,j}^i), \;\; e_{\beta_{c,j}^i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\beta_{c,j}^i) \text{ for } j \neq i$$

4. Each player runs Π_{ZKPoPK} acting as a prover for the encryptions he generated. The protocol aborts if any proof fails.
5. All players now locally compute

$$\begin{aligned} e_a &\leftarrow e_{a,1} \boxplus \cdots \boxplus e_{a,n} = \mathsf{Enc}_{\mathsf{pk}}(a_1 + \cdots + a_n), \\ e_b &\leftarrow e_{b,1} \boxplus \cdots \boxplus e_{b,n} = \mathsf{Enc}_{\mathsf{pk}}(b_1 + \cdots + b_n), \\ e_r &\leftarrow e_{r,1} \boxplus \cdots \boxplus e_{r,n} = \mathsf{Enc}_{\mathsf{pk}}(r_1 + \cdots + r_n), \\ e_{r+c} &\leftarrow e_r \boxplus (e_a \boxtimes e_b)\,. \end{aligned}$$

6. The players invoke $\mathcal{F}_{\text{KeyGenDec}}$ to decrypt e_{r+c} and thereby obtain $r + c$.
7. The players now set $e_{c,1}$ to be $e_{c,1} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathsf{encode}(r + c), 0^d) \boxminus e_{r,1}$, and set, for $i \neq 1$, $e_{c,i} \leftarrow \boxminus e_{r,i}$.
8. Each player sets $c_i \leftarrow -r_i$, except for player one who sets $c_1 \leftarrow (r + c) - r_1$. Notice that all players now have encryptions $e_{c,i}$ of c_i and that

$$c_1 + \cdots + c_n = ((c + r) - r_1) - r_2 - \cdots - r_n = c + (r - r) = c.$$

9. To compute the MAC's on each data item a_j (and similarly on b_j, c_j) all the players compute $e_{\mathsf{MAC}^i(a_j)} \leftarrow e_{a,j} \boxtimes e_{\alpha_j^i} \boxplus e_{\beta_{a,j}^i}$. Then they invoke $\mathcal{F}_{\text{KeyGenDec}}$ to decrypt to party j, who will then get $\mathsf{MAC}^i(a_j)$.

Fig. 4. The protocol for constructing an unreliable triple.

4.3 An improved online phase

In this section we construct a new online phase for the MPC protocol. Before presenting the concrete protocol we give the intuition and motivation behind the construction.

Protocol Π_{ONLINE}

Initialize: The parties first invoke the preprocessing to get the shared secret key $[\alpha]$ and a sufficient number of multiplication triples $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$ and single values $\langle r\rangle$.

Input: To share P_i's input x_i, P_i takes an available single value $\langle r\rangle$. Then, the following is performed:

1. $\langle r\rangle$ is partially opened to P_i.
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 $\langle a\rangle, \langle b\rangle$, the parties locally compute $\langle a\rangle + \langle b\rangle$.

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

1. They take a triple $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$ from the set of the available ones.
2. $\rho := \langle x\rangle - \langle a\rangle$ and $\sigma := \langle y\rangle - \langle b\rangle$ are partially opened.
3. They compute $\langle z\rangle \leftarrow \langle c\rangle + \rho\langle b\rangle + \sigma\langle a\rangle + \rho\sigma$
4. They remove $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$ from the set of the available triples.

Output: We enter this stage when the players have $\langle y\rangle$ for every output value y, but no such value has been opened (the output values are only correct if players have behaved honestly). We then do the following:

1. Let $a_1, \ldots, a_T$ be all values 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 set of public values $e_1, \ldots, e_T \in \mathbb{F}_{p^k}$ are chosen. We give details below on how this is done. 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 each 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 so α is now public.
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, players conclude that the output vales are correctly computed.
5. To output $\langle y\rangle$ to P_i, $\langle y\rangle$ is opened to P_i, i.e., the commitments to $y_i, \gamma(y)_i$ are opened to P_i. Now, y is defined as $y := \sum_i y_i$ and P_i checks that $\alpha(y + \delta) = \sum_i \gamma(y)_i$, if so, y is his output.

Fig. 5. The improved online phase.

To execute the online phase of [3], each player has to store a key and MAC for every other player. As a result the total amount of data players need to store is quadratic in n. This seems unavoidable if one uses unconditional authentication to authenticate each share of each secret value. Here, we therefore use a different idea, based on the observation that what we really want to authenticate is the shared value, not the shares. 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 share a MAC αa (note that this is slightly different from our earlier construction). This way to represent secret values is linear, just like the $[\cdot]$-representation, and we can therefore do secure multiplication based on multiplication triples we produce in the preprocessing.

An immediate problem with this is that opening a value 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 [3]. We resolve this by opening only the shared values and keeping all information on MACs secret until we are ready to open the outputs (of course, this may mean that some of the opened values are incorrect). At this point, we form a random linear combination of all opened values to get a single value a, and players compute the same linear combination of their shares of the corresponding MACs. This results in shares of a (hopefully) valid MAC on a. All players now commit to these shares, as well as their shares of the output. Now we reveal α, and then the shares of the MAC on a are revealed so we can check. The intuition is that, because of the commitments, when α is revealed it is too late for corrupt players to exploit knowledge of the key. Therefore, if the MAC checks out, all opened values were correct with high probability, so we can trust that the output values we computed are correct and can safely open them.

Representation of values and MACs. We assume the preprocessing outputs $[\alpha]$, where $[\cdot]$ is the representation we defined above, and α is fixed for the entire online phase. Then each shared value $a \in \mathbb{F}_{p^k}$ is represented as

[3] Checking using secure computation while keeping α secret will not work and only leads to circularity: secure computing is what we are trying to do in the first place.

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 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 and we can do multiplications in the standard way if the preprocessing outputs random triples $\langle a \rangle, \langle b \rangle, \langle c \rangle$ with $ab = c$. However, in our protocol we are not guaranteed the correct result since we do not immediately check the MACs of the opened values. As sketched above we postpone the checking to the end of the protocol in the output phase. During 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 broadcasts a_i, and all parties output $a = a_1 + \cdots + a_n$.

The full protocol for the online phase is shown in Figure 5. 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 on request from the committer. Such a functionality can be implemented efficiently based, e.g., on Paillier encryption or the DDH assumption [9, 14]. However, we actually do not need an extra assumption. We show in Appendix B.4 that we can do ideal commitments based only on $\mathcal{F}_{\text{TRIP}}$.

Preprocessing. Based on the preprocessing phase we described earlier it is very easy to design a preprocessing that will produce representations as defined here: first build $[\alpha]$ for a random α. Then build as many values $[a]$ for random a as we require. Use a multiplication triple to compute $[\alpha a]$. The players now form $\langle a \rangle$ as follows: use the shares of a from $[a]$ as $a_1, \ldots, a_n$, use the shares of αa from $[\alpha a]$ as $\gamma(a)_1, \ldots, \gamma(a)_n$ and set $\delta = 0$. We emphasize that, while easy to explain, this method is not the most efficient way to build such representations, it can be done more efficiently by using the SHE scheme in a more direct way. We will for simplicity assume we have an ideal functionality for the preprocessing $\mathcal{F}_{\text{PREP}}$ that will create the necessary representations needed for the online protocol.

Generating the e_i's. In Appendix B.4, we discuss various ways to generate the e_i. The most efficient solution is to have a random value shared $[x]$. In the output stage, x is opened and the e_i's are computed as $e_i \leftarrow x^i$. This has the advantage of adding only a constant number of multiplications in $\mathbb{F}_{p^k}$ for a secure multiplication. The probability to cheat in the output stage with this solution is $(T + 1)/p^k$ where T is the number of values opened during the protocol.

Complexity. As for the complexity of this online protocol, note that we only use commitments to commit to shares of a single MAC and to shares of the output. This cost is in general independent of the size of the circuit we compute. The dominating cost is therefore the cost of a secure multiplication. Using the method for generating the e_i where $e_i \leftarrow x^i$, the cost is easily seen to be $O(n)$ multiplications in $\mathbb{F}_{p^k}$ and $O(n^2)$ additions.

We can now state the new theorem on security of the online phase, the proof is in Appendix B.4.

Theorem 4. *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 active adversary corrupting up to $n - 1$ parties.*

5 Concrete Instantiation of the Abstract Scheme based on LWE

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

We first focus on the basic scheme before moving on to the version with distributed decryption:

$\underline{\mathsf{ParamGen}(1^\kappa, M)}$: 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 $l' \geq l$ 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 be using the public key version of the Brakerski–Vaikuntanathan scheme [5]. Given the above set up, key generation proceeds as follows:

- $\mathbf{a} \leftarrow A_q$ and $\mathbf{s}, \mathbf{e} \leftarrow D_{\mathbb{Z}^N,s}$, treating $\mathbf{s}$ and $\mathbf{e}$ as elements of A_q
- $\mathbf{b} \leftarrow (\mathbf{a} \cdot \mathbf{s}) + (p \cdot \mathbf{e})$.
- Return $\mathsf{pk} \leftarrow (\mathbf{a}, \mathbf{b})$ 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 $\mathbf{r} \in \mathsf{D}_p^d$, we proceed as follows.

- Parse $\mathbf{r}$ as $(\mathbf{u}, \mathbf{v}, \mathbf{w}) \in (\mathbb{Z}^N)^3$.
- $\mathbf{c}_0 \leftarrow (\mathbf{b} \cdot \mathbf{v}) + (p \cdot \mathbf{w}) + \mathbf{x}$ and $\mathbf{c}_1 \leftarrow (\mathbf{a} \cdot \mathbf{v}) + (p \cdot \mathbf{u})$.
- Return $(\mathbf{c}_0, \mathbf{c}_1, 0)$.

$\underline{\mathsf{Dec}_{\mathsf{sk}}(c)}$: 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. 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 executes the following steps

- $\widehat{\mathbf{a}}, \widehat{\mathbf{b}} \leftarrow A_q$
- Return $\widehat{\mathsf{pk}} := (\widehat{\mathbf{a}}, \widehat{\mathbf{b}})$.

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

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

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

As remarked earlier this one-time setup procedure can be accomplished by standard UC-secure multiparty computation protocols such as that described in [3]. The following theorem is proved in Appendix B.5. It depends on the constant B defined above. In Appendix C 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 5. *In the $\mathcal{F}_{\text{KEYGEN}}$-hybrid model, the protocol Π_{DDEC} (Figure 6) 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. 6. The distributed decryption protocol.

6 Acknowledgements

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 thereon. 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 the presence of errors. Manuscript, 2011.
2. D. Beaver. Efficient multiparty protocols using circuit randomization. *Advances in Cryptology – Crypto '91*, Springer LNCS 576, 420–432, 1991.
3. R. Bendlin, I. Damgård, C. Orlandi and S. Zakarias. Semi-homomorphic encryption and multiparty computation. *Advanced in Cryptology – EuroCrypt 2011*, Springer LNCS 6632, 169–188, 2011.
4. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. IACR e-print 2011/277.
5. Z. Brakerski and V. Vaikuntanathan. Fully homormorphic encryption from ring-LWE and security for key dependent messages. *Advances in Cryptology – Crypto 2011*, Springer LNCS 6841, 505–524, 2011.
6. Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. IACR e-print 2011/344.
7. J.B. Nielsen, P.S. Nordholt, C. Orlandi and S.S. Burra. A new approach to practical active-secure two-party computation. Cryptology ePrint Archive, Report 2011/091.
8. R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally composable two-party and multi-party secure computation. *Symposium on Theory of Computing – STOC 2002*, 494–503, ACM,2002.
9. I. Damgård and J.B. Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. *Advances in Cryptology – Crypto 2002*, Springer LNCS 2442, 581–596, 2002.
10. I. Damgård and C. Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. *Advances in Cryptology – Crypto 2010*, Springer LNCS 6223, 558–576, 2010
11. N. Gama and P.Q. Nguyen. Predicting lattice reduction. *Advances in Cryptology – Eurocrypt 2008*, Springer LNCS 4965, 31–51, 2008.
12. C. Gentry. Fully homomorphic encryption using ideal lattices. *Symposium on Theory of Computing – STOC 2009*, ACM, 169–178, 2009.
13. K. Lauter, M. Naehrig and V. Vaikuntanathan. Can homomorphic encryption be practical? IACR e-print 2011/405.
14. Y. Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. *Advances in Cryptology – Eurocrypt 2011*, Springer LNCS 6632, 446–466, 2011.
15. V. Lyubashevsky, C. Peikert and O. Regev. On ideal lattices and learning with errors over rings. *Full version*, Manuscript, 2011.
16. R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. *Topics in Cryptology – CT-RSA 2011*, Springer LNCS 6558, 319-339, 2011.
17. D. Micciancio and O. Regev. Lattice-based cryptography. *Post-Quantum Cryptography*, 147–192, 2009.
18. S. Myers, M. Sergi and a. shelat. Threshold fully homomorphic encryption and secure computation. Cryptology ePrint Archive, Report 2011/454.
19. M. Püschel and J.M.F. Moura. Algebraic signal processing theory: Cooley–Tukey type algorithms for DCTs and DSTs. *IEEE Trans. Signal Processing*, **56**, 1502–1521, 2008.
20. N.P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. IACR e-print 2011/133.

A 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.

A.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.

A.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 6. *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 [19] and the full version of [15]. 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 6). 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

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

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:

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

Let m' be a divisor of m then for $i \in \{0, \ldots, m-1\}$ 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

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

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 $0 \le i < m-1$, to row $i \cdot m' \bmod m - 1$. Following [19] 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

$$\mathsf{DFT}_m = L^m_{m'} \cdot (I_k \otimes \mathsf{DFT}_{m'}) \cdot T_{m,m'} \cdot (\mathsf{DFT}_k \otimes I_{m'}), \tag{1}$$

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

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

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

Lemma 1. *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

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

□

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 2. *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 [15] the authors specialise the decomposition (1) (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

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

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

Lemma 3. *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 4. *If p is a prime then*

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

Proof. [4] 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

$$\frac{\Phi_p(X)}{\Phi'_p(\zeta_p) \cdot (X - \zeta_p)}, \tag{2}$$

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 (2) 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.

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

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.

A.3 Application of the above bounds

An immediate consequence of Theorem 6 is to provide an upper bound on the value δ_K 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 5. *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

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

B Proofs

B.1 Zero-Knowledge Proof

Construction of the Protocol. 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 generates $\mathbf{y}_i \leftarrow \mathbb{Z}^N$ and $\mathbf{s}_i \leftarrow \mathbb{Z}^d$, such that $\|\mathbf{y}_i\|_\infty \leq 2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \tau$ and $\|\mathbf{s}_i\|_\infty \leq 2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \rho$.
- 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 $T = S + M_e \cdot R$. He returns $(\mathbf{z}, T)$ 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 whether the following three conditions hold

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

Fig. 7. The ZKPoPK Protocol.

Proof (Theorem 1).

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:

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

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 $M_{\mathbf{e},i} \cdot \mathbf{x}^\mathsf{T} \leq \mathsf{sec} \cdot \tau$. This means that $\|\mathbf{y}_i\|_\infty$ is taken from an exponentially bigger interval than $M_{\mathbf{e},i} \cdot \mathbf{x}^\mathsf{T}$. Therefore the check $\|\mathbf{z}_i\|_\infty \leq 2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \tau$ fails only with negligible probability. A similar argument shows that the check $\|\mathbf{t}_i\|_\infty \leq 2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \rho$ fails only with negligible probability.

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

$$(M_\mathbf{e} - M_{\mathbf{e}'}) \boxtimes \mathbf{c}^\mathsf{T} = \left(\mathbf{d} \boxminus \mathbf{d}'\right)^\mathsf{T} \tag{3}$$

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 (3) 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 $c_i = \mathsf{Enc}_\mathsf{pk}(\mathbf{x}_i, \mathbf{r}_i)$ for $i = 1, \ldots, \mathsf{sec}$, it is possible for the verifier to directly solve the linear system in $\mathbf{x}$ and R (since the cryptosystem is additively homomorphic, from the bottom equation to the top one.
Since $\|\mathbf{z}_i\|_\infty \leq 2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \tau$, $\|\mathbf{t}_i\|_\infty \leq 2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \rho$, $\|\mathbf{z}'_i\|_\infty \leq 2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \tau$ and $\|\mathbf{t}'_i\|_\infty \leq 2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \rho$, then $c_{\mathsf{sec}-i}$ is a $(2^{\mathsf{sec}+i+\log \mathsf{sec}} \cdot \tau, 2^{\mathsf{sec}+i+\log \mathsf{sec}} \cdot \rho)$-ciphertext (by induction on i). Therefore, $\mathbf{c}$ contains $(2^{2\mathsf{sec}+\log \mathsf{sec}} \cdot \tau, 2^{2\mathsf{sec}+\log \mathsf{sec}} \cdot \rho)$-ciphertexts.

Zero-Knowledge: We give an honest-verifier simulator for the protocol that outputs accepting conversations. In order to do that, the simulator samples $\mathbf{e} \in \{0, 1\}^\mathsf{sec}$ uniformly and $\mathbf{z}, T$ uniformly with the constrain that $\mathbf{d}$ contains $(2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \tau, 2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \rho)$-ciphertexts. Finally, $\mathbf{a}$ is computed as $\mathbf{a}^\mathsf{T} \leftarrow \mathbf{d}^\mathsf{T} \boxminus (M_\mathbf{e} \cdot \mathbf{c}^\mathsf{T})$. The output of the simulator is $(\mathbf{a}, \mathbf{e}, (\mathbf{z}, T))$.
By construction we have $\mathbf{d}^\mathsf{T} = \mathbf{a}^\mathsf{T} \boxplus (M_\mathbf{e} \cdot \mathbf{c}^\mathsf{T})$. Moreover, the distribution of a simulated $\mathbf{e}$ is the same as a real one. In a real conversation $\mathbf{a}$ consists of uniform $(2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \tau, 2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \rho)$-ciphertexts. In a simulated conversation $\mathbf{a}$ is computed as a function of $\mathbf{d}$ (containing uniform $(2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \tau, 2^{\mathsf{sec}-1+\log \mathsf{sec}} \cdot \rho)$-ciphertexts) and $\mathbf{c}$ (containing uniform (τ, ρ)-ciphertexts). Since $M_\mathbf{e}$ contains only zeros and ones, $M_\mathbf{e} \cdot \mathbf{c}^\mathsf{T}$ contains $(\mathsf{sec} \cdot \tau, \mathsf{sec} \cdot \rho)$-ciphertexts. Thus, since the contribution of $M_\mathbf{e} \cdot \mathbf{c}^\mathsf{T}$ on a simulated $\mathbf{a}$ is exponentially smaller than the one given by $\mathbf{d}$, we have that simulated $\mathbf{a}$'s are statistically close to real $\mathbf{a}$'s. A similar argument can be made for the statistically indistinguishability of simulated $(\mathbf{z}, T)$ and real $(\mathbf{z}, T)$. □

B.2 Online Phase

Proof (Theorem 2).

We construct a simulator $\mathcal{S}_{\text{AMPC}}$ such that a poly-time environment $\mathcal{Z}$ cannot distinguish between the real protocol system $\mathcal{F}_{\text{TRIP}}$ composed with Π_{AMPC} and $\mathcal{F}_{\text{AMPC}}$ composed with $\mathcal{S}_{\text{AMPC}}$. We assume here static, active corruption. The simulator will internally run a copy of $\mathcal{F}_{\text{TRIP}}$ composed with Π_{AMPC} where it corrupts the parties specified by $\mathcal{Z}$. The simulator relays messages between parties/$\mathcal{F}_{\text{TRIP}}$ and $\mathcal{Z}$, such that $\mathcal{Z}$ will see the same interface as when interacting with a real protocol. During the run of the internal protocol the simulator will keep copies of the shares, MACs and keys of both honest and corrupted parties and update them according to the execution.

The idea is now, that the simulator runs the protocol with the environment, where it plays the role of the honest parties. Since the inputs of the honest parties are not known to the simulator these will be random values (or zero). However, if the protocol is secure, the environment will not be able to tell the difference. The specification of the simulator $\mathcal{S}_{\text{AMPC}}$ is presented in Figure 8.

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

Initialize: The simulator initializes the copy with p and creates the desired number of triples by doing the steps in $\mathcal{F}_{\text{TRIP}}$. Note that here the simulator will read all data (shares, keys, errors) of the corrupted parties specified to the copy of $\mathcal{F}_{\text{TRIP}}$.

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 in Step 1 during input, the MACs are not correct, the protocol is aborted.

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 $x_i' = a + \delta \mod p$ since it knows (all the shares of) a. 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, as before, aborts if some MAC is not correct. Otherwise it calls $multiply$ on the ideal functionality $\mathcal{F}_{\text{AMPC}}$.

Output: If P_i is honest the output step is run and the protocol is aborted if some MAC is not correct. Otherwise the simulator calls $output$ on $\mathcal{F}_{\text{AMPC}}$.

If P_i is corrupted the simulator calls $output$ on $\mathcal{F}_{\text{AMPC}}$. Since P_i is corrupted the ideal functionality will provide the simulator with y, which is the output to P_i. Now it has to simulate shares y_j of honest parties such that they are consistent with y. This is done by changing one of the internal shares of an honest party. Let P_k be that party. The new share is now computed as $y_k' \leftarrow y - \sum_{i \neq k} y_i$. Next, the MAC for y_k' is needed. This, the simulator can compute from scratch as $\mathsf{MAC}_{K^i_{y_k}}(y_k')$ since it knows from the beginning the keys of P_i. This enables it to compute $K^i_{y_k}$ by the computations on representations done during the protocol. Before showing the MAC to the environment, the simulator has to add the error $\Delta^i_{y_k}$ which is due to the errors introduced to the MACs during initialization. As for $K^i_{y_k}$, the simulator can compute $\Delta^i_{y_k}$ as a linear combination corresponding to the computations on representations done during the protocol. Now the simulator sends the internal shares and (possibly wrong) MACs to P_i.

Fig. 8. The simulator for $\mathcal{F}_{\text{AMPC}}$.

In general, the openings in the protocol do not reveal any information about the values that the parties have. Whenever we open a value during Input or Multiply we mask it by subtracting with a new random value. Therefore, the distribution of the view of the corrupted parties is exactly the same in the simulation as in the real case. Then, the only method there is left for the environment to distinguish between the two cases, is to compare the protocol execution with the inputs and outputs of the honest parties and check for inconsistency.

If the simulated protocol fails at some point because of a wrong MAC, the simulator aborts which is consistent with the internal state of the ideal functionality since, in this case, the simulator also makes the ideal functionality fail.

If the simulated protocol succeeds, the ideal functionality is always told to output the result of the function evaluation. This result is of course the correct evaluation of the input matching the shares that were read from the corrupted parties in the beginning. Therefore, if the corrupted parties during the protocol successfully cheat with their shares, this would not be consistent. However, the probability of a party being able to claim a wrong value for a given MAC is $1/p^k$. In conclusion, if the protocol succeeds, the computation is correct except with probability a polynomial multiple (the number of MACs ever checked) of $1/p^k$. □

B.3 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 $\mathsf{Enc}_{\widetilde{\mathsf{pk}}}(x)$ 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 $\mathcal{S}_{\text{TRIP}}$ for Π_{TRIP} that will run a copy of the protocol internally and simulate the calls to $\mathcal{F}_{\text{KEYGEN}}$ and $\mathcal{F}_{\text{RAND}}$. During the protocol execution the simulator plays the honest players' part. Here it

obtains the values of the corrupted players, since it knows the secret key. These values are then input to $\mathcal{F}_{\text{TRIP}}$. A precise description is provided in Figure 9.

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

Initialize: The simulator first performs the initialization steps of Π_{TRIP}. To simulate the call to $\mathcal{F}_{\text{KeyGenDec}}$ in step 1, it uses $\mathsf{KeyGen}()$ to generate the key pair $(\mathsf{sk}, \mathsf{pk})$. It sends pk to all players and stores sk. Then, it does steps 2-4 where it aborts if an invocation of Π_{ZKPoPK} fails.
Finally it calls Initialize on $\mathcal{F}_{\text{TRIP}}$ where it inputs α_j^i for each pair P_i, P_j where P_i is corrupted and P_j is honest. The simulator obtains these by decrypting the values that are broadcast in step 3.

Triple: The simulator again performs the steps of Π_{TRIP}. During this, Π_{ZKPoPK} is invoked a number of times. If any invocation fails, the simulator will abort. When doing step 6 and 9 of Π_{TRIP}, it will again simulate calls to $\mathcal{F}_{\text{KeyGenDec}}$ by performing exactly what is described in step 2 and 3 of $\mathcal{F}_{\text{KeyGenDec}}$, respectively. Then, in the final step of Π_{TRIP} the check on a triple is done, where again the simulator will abort, if the check fails.
Finally the simulator calls Triple on $\mathcal{F}_{\text{TRIP}}$ where it inputs the shares of the corrupted players. The simulator obtains the shares by decrypting the values that are broadcast in step 3 of Π_{TRIP}.

Fig. 9. The simulator for $\mathcal{F}_{\text{TRIP}}$.

Security is argued by a reduction similar to the one in [3]: We assume there exists a PPT algorithm D that can distinguish with significant probability between a real and a simulated view. Then, we can use D to distinguish between a normally generated public key and a meaningless one with basically the same success probability as D. This leads to a contradiction, since a key generated by the normal key generator is computationally indistinguishable from a meaningless one.

In order to do so, we construct an algorithm B that takes as input a public key pk^* (either a normal public key or a meaningless one) and outputs a view that can be input to D. 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, then if the bit is set to "Real", B's output is indistinguishable from a real execution of the protocol, while if the bit is set to "Simulation", B's output is indistinguishable from a simulated run. However, in case pk^* is meaningless, both choices of the bit lead to an output indistinguishable from a simulated run. Then, $\widetilde{B}$'s output is used as input to D. If D guesses correctly, D' guesses that $\mathsf{pk}^* = \mathsf{pk}$ otherwise it guesses $\mathsf{pk}^* = \widetilde{\mathsf{pk}}$.

For simplicity we describe in the following, the algorithm B for the two-party setting, where we have a corrupt party P_1 and an honest party P_2: We start by letting the input pk^* be the public key pk. After this, we begin executing the protocol. However, during step 6 and 9 of Π_{TRIP}, we cannot decrypt since we do not know the secret key. Instead, we exploit that P_1 and P_2 have run in step 4 of the protocol Π_{ZKPoPK} with P_1 as prover. That is, P_1 has proved that he knows encodings of appropriate size corresponding to the plaintext inside the ciphertexts broadcast in step 3. This means we can use the knowledge extractor of the protocol followed by decoding to extract the shares from P_1. With these we calculate the resulting plaintext ($r + c$ in step 6 and $\alpha_j^i \cdot a_j + \beta_{a,j}^i$ in step 9). When simulating decryption to P_2, P_1 will provide an error δ that should be added to the plaintext. Now, we continue the protocol as if we had decrypted. In the end we choose randomly to generate a real or a simulated view. In the first case, we let the output be exactly those values that were used in our execution of the protocol. That is, all shared values are determined from the shares we used, so for example $a = a_{P_1}^{Real} + a_{P_2}^{Real}$. In the second case, we choose the output for P_2 as $\mathcal{F}_{\text{TRIP}}$ would do. That means the shares of a given triple $a_{P_1}, b_{P_1}, c_{P_1}$ will now be determined by choosing a, b at random, setting $c \leftarrow a \cdot b$ and then letting $a_{P_2} \leftarrow a - a_{P_1}^{Real}, b_{P_2} \leftarrow b - b_{P_1}^{Real}, c_{P_2} \leftarrow c - c_{P_1}^{Real}$.

It can now be seen that if pk^* was 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 difference to the actual simulator is that the simulator opens the ciphertexts in steps 3, 6 and 9 using the secret key to decrypt, while B uses the extractor for Π_{ZKPoPK} in step 4 and computes the plaintexts from its results. However, for every ciphertext c generated by P_1, the extractor for Π_{ZKPoPK} will, except with negligible probability, be able to find

an encoding $\mathbf{x}$ (resp. randomness $\mathbf{r}$) smaller than B_{plain} (resp. B_{rand}), with $c = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}, \mathbf{r})$. This follows from soundness of Π_{ZKPoPK} and admissibility of the cryptosystem. Then, by correctness of the cryptosystem, computing the plaintexts as B does 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.

On the other hand if pk^* is a meaningless key, we know first of all that the encryptions contain statistically no information about the values. In addition, 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. This is indeed the case, since the zero-knowledge property implies that the conversations could just as well have been simulated without knowing what is inside the encryptions. We conclude that up to and including step 6, essentially no information on any value held by P_2 is revealed.

It follows that step 6 reveals no information on c since $r + c$ is just a random value. In the last step of Π_{TRIP}, we have the opening and check of triples. Here, no information on a, b is leaked from $ea - f$, and $b - g$, since these are just random values.

We conclude that the view of the corrupt player(s) is essentially independent of the internal values B uses to generate the output, and hence also independent of whether B chooses the real or the simulated case. Therefore, the only chance D has of distinguishing is to consider also the output generated for the honest player(s). If the protocol aborts no such output is generated so we need only to consider views where the protocol completes. In the simulated case, by construction the shares of P_1 and P_2 together always determine for each triple random values a, b, c with $ab = c$. The same is true in the real case with overwhelming probability, and this follows from the check on the triples: 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, the probability of satisfying the check is essentially $1/p^k$: first, if P_2 does not reveal the shares he is committed to, the check fails with overwhelming probability since P_2 has (essentially) no information on the key P_1 uses to check the MACs. On the other hand, if P_1 sends correct shares the check fails except with probability $1/p^k$, since there is only one random challenge e, for which $e \cdot (c - a \cdot b) = (h - g \cdot f)$. Therefore, the event that the protocol does not abort but some triple is incorrect happens with negligible probability, assuming $p^k = \Omega(2^{\mathsf{sec}})$.

As a result, if we use a meaningless key, a real execution and a simulated execution are statistically indistinguishable. Therefore, if D can distinguish real views from simulated views with some advantage ϵ, then D' can distinguish normal keys from meaningless keys with advantage $\epsilon - \delta$. We subtract δ, since there is some negligible error that B does not succeed, for instance if the adversary is able to cheat with the check of the triples. However, if ϵ is non-negligible, then $\epsilon - \delta$ is also non-negligible, and so we have our contradiction. □

B.4 New Online Phase

Proof (Theorem 4).

Before proving the online protocol UC secure, we argue what is the probability getting away with cheating in step 4 of 'Output'. 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 the values they open during the protocol execution. Also, since α, γ are secret shared in the protocol, the information held by the adversary in the security game is the same.

Now, let us look at the probability of winning the game if the e_i 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 discuss alternative ways to generate them. An approach would be to use a pseudorandom generator G. We would then have shared some random seed $[s]$. By opening $[s]$ and feeding it to G we can generate T pseudorandom elements. With respect to cheating the argument is basically the same; If the adversary 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 this adversary to break G. With this way of generating the e_i's, we increase the complexity for one secure multiplication with whatever G needs to generate one pseudorandom element.

An alternative way that does not increase the complexity significantly is to again have a random value shared $[x]$. Now, however, the e_i's are computed as $e_i \leftarrow x^i$. This has the advantage of adding only a constant number of multiplications in $\mathbb{F}_{p^k}$ for a secure multiplication, leaving the total asymptotic complexity unchanged. On the security side, we need in this case to choose the field size more carefully. We still want that $\sum_{i=0}^{T} x^i \mu_i = 0$ should happen with small probability. Viewing f_μ as a polynomial of degree T, we know it has T roots, so we have to make sure we have an upper bound on T such that x is chosen from a field big enough for T/p^k to be small.

We now 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/$\mathcal{F}_{\text{TRIP}}$ 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 10.

In general, the openings in the protocol do not reveal any information about the values that the parties have. Whenever we partially open a value during Input or Multiply we mask it by subtracting with a new random value. Therefore, the distribution of the view of the corrupted parties is exactly the same in the simulation as in the real case. Then, the only method there is left for the environment to distinguish between the two cases, is to compare the protocol execution with the inputs and outputs of the honest parties and check for inconsistency.

If the simulated protocol fails at some point because of a wrong MAC, the simulator aborts which is consistent with the internal state of the ideal functionality since, in this case, the simulator also makes the ideal functionality fail. If the simulated protocol succeeds, the ideal functionality is always told to output the result of the function evaluation. This result is of course the correct evaluation of the input matching the shares that were read from the corrupted parties in the beginning. Therefore, if the corrupted parties during the protocol successfully cheat with their shares, this would not be consistent. The shares are checked in step 4 and 5 of 'Output'. As we discussed above, the probability of a party being able to cheat in step 4 is $2/|\mathbb{F}_{p^k}|$, $2/|\mathbb{F}_{p^k}|+$ negligible or $T/|\mathbb{F}_{p^k}|$, depending on the way we generate the e_i's. In step 5, only one MAC is checked for

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 $x'_i \leftarrow (r + \delta) \mod p$ 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: If P_i is honest the output step is run and the protocol is aborted if one of the checks in step 4 or 5 does not go through. Otherwise the simulator calls *output* on $\mathcal{F}_{\text{AMPC}}$.

If P_i is corrupted the simulator calls *output* on $\mathcal{F}_{\text{AMPC}}$. Since P_i is corrupted the ideal functionality will provide the simulator with y, which is the output to P_i. Now it has to simulate shares y_j of honest parties such that they are consistent with y. This is done by changing one of the internal shares of an honest party. Let P_k be that party. The new share is now computed as $y'_k \leftarrow y - \sum_{i \neq k} y_i$. Next, the MAC for y'_k is needed. This, the simulator can compute as $\gamma'_k(y'_k) \leftarrow \alpha(y'_k + \delta)$ since it knows from the beginning (all the shares of) α. Now it simulates the openings of $y'_k, \gamma'_k(y'_k)$ towards the environment.

Fig. 10. The simulator for $\mathcal{F}_{\text{AMPC}}$.

each output, so here the probability of cheating is $1/|\mathbb{F}_{p^k}|$ as argued earlier. In conclusion, if the protocol succeeds (and assuming a polynomial number of outputs), the computation is correct except with probability a polynomial multiple of $1/|\mathbb{F}_{p^k}|$. □

Commitments based on $\mathcal{F}_{\text{TRIP}}$ 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 $\mathcal{F}_{\text{TRIP}}$ 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 $[x]$ and recomputes the MACs to make it fit with the broadcasted value and the value he should have committed to.

B.5 Distributed Decryption

Proof (Theorem 5). 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.

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 analyze 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.

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. 11. The simulator for Π_{DDEC}.

Hybrid: On input $(\mathsf{sk}_i)_{i=1,\ldots,n}, c$, 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 $2^{\mathsf{sec}} \cdot B$; moreover, since $\mathbf{t} - \mathsf{encode}(m)$ is a multiple of p, one can write $\widetilde{\mathbf{t}}_j = \varphi(U + (\mathsf{encode}(m) - \mathbf{t})/p)$. Since $\|(\mathsf{encode}(m) - \mathbf{t})/p\|_\infty \leq (B+1)/p$ and U is uniform in an exponentially larger range, then the distribution $U + (\mathsf{encode}(m) - \mathbf{t})/p$ is statistically close to U. 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. □

C 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 $B_{plain} \geq \tau 2^{2\mathsf{sec}+\log \mathsf{sec}}$, $B_{rand} \geq \rho 2^{2\mathsf{sec}+\log \mathsf{sec}}$, and where C, the set of functions we can evaluate on ciphertexts, contains all formulas evaluated in the protocol Π_{TRIP}, (including the identity function).

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 [5], 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 noise 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 [17][Section 5.4.1], we require

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

Following the work of [11] 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 (where δ is a function of the security parameter sec, typically one may select $\delta = 1.005$) 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 [17] 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 "error term" 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 A.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 $B_{plain} \geq \tau 2^{2\mathsf{sec}+\log \mathsf{sec}}$ and $B_{rand} \geq \rho 2^{2\mathsf{sec}+\log \mathsf{sec}}$, where 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. $\rho \approx Y$.

Since $B_{plain} \ll B_{rand}$ we therefore need to ensure correctness, for our explicit circuits, for ciphertexts whose noise is bounded by $Y' = Y \cdot 2^{\mathsf{sec}+\log \mathsf{sec}+1}$. In our MPC protocol we 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 A.3 crudely estimate the size of B, from Section 5, needed to ensure valid decryption. Our crude (over-) estimate therefore come 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 $c_{\mathsf{sec}} = 2^{\mathsf{sec}+\log \mathsf{sec}+1}$. 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 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

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

where sec is the security parameter, δ is a function of sec 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 $\delta = 1.005$.

Example 1. We first look at $p > 2^{32}$. Our first (approximate) search reveals we need $N > 13200$, $q \approx 2^{380}$ 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 $N = \phi(m) > 13200$. A quick search reveals candidates of

$$(p, m, N) = (2^{32} + 163,\ 13337,\ 13336).$$

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 4, 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 13337 in addition allows us to evaluate $s = p - 1 = 13336$ 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

$$\frac{N \cdot \log_2(q)}{N \cdot \log_2(p)} = \frac{\log_2(q)}{\log_2(p)} = \frac{380}{32} \approx 11.87.$$

Example 2. Performing the same analysis for a $p > 2^{64}$, our first naive search of parameters reveal we need an $n \approx 15600$ and $q \approx 2^{450}$. We then search for specific parameters and find $p = 2^{64} + 1945$ is pretty near to optimum, which results in a prime value of m of 16111. We find the expansion factor is given by

$$\frac{\log_2(q)}{\log_2(p)} = \frac{450}{64} \approx 7.03.$$

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 $N > 11100$, $q \approx 2^{320}$ and $r = 3.2$. So we now need to determine a value m such that

$$N = \phi(m) > 11100 \text{ and } 2^d \equiv 1 \pmod{m} \text{ and } d \equiv 0 \pmod{8}.$$

A quick search reveals candidates of

$$(m, N) = (15665, 11520)$$

since $\Phi_{15665}(X)$ factors into $s = 480$ factors of degree $d = 24$ modulo 2. Thus using this value of m we are able to work with $s = 480$ elements of $\mathbb{F}_{2^8}$ in parallel. The message expansion factor, given we require $N \cdot \log_2(q)$ bits to represent 480 elements in $\mathbb{F}_{2^8}$ is given by

$$\frac{N \cdot \log_2(q)}{8 \cdot s} = \frac{d \cdot 480}{8} = 1440.0$$

For this value of m we find $C_{15665} \approx 14.916$.

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 Xeon E5620 running at 2.40 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	0.98 {0.00007}	0.49 {0.00004}	1.98 {0.0001}	0.99 {0.00007}
2	1.27 {0.00008}	0.65 {0.00004}	2.56 {0.0001}	1.26 {0.00008}
3	0.83 {0.00173}	0.43 {0.00089}	1.69 {0.0035}	0.83 {0.00173}

D 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. 12. The ideal functionality for coin-flipping.

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

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

Rand: On input $(rand, P_i, varid)$ from all parties P_i, with $varid$ a fresh identifier, the functionality picks $r \leftarrow \mathbb{F}_{p^k}$ and stores $(varid, r)$.

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 $(output, P_i, varid)$ from all parties (if $varid$ is present in memory), the functionality retrieves $(varid, x)$ and outputs it to P_i.

Fig. 13. The ideal functionality for arithmetic MPC.

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

Initialize: On input $(init, p)$ from all parties the functionality stores the modulus p. For each corrupted party P_i the environment specifies values $\alpha_j^i, j = 1, \ldots, n$, except those α_j^i where both P_i and P_j are corrupt. For each honest P_i, it chooses $\alpha_j^i, j = 1, \ldots, n$ at random.

Triple: On input $(triple)$ from all parties P_i, the functionality does the following:

1. It waits to get from the environment either "stop", or some data as specified below. In the first case it sends "fail" to all honest parties and stops. In the second case, the environment specifies for each corrupt party P_i, a share a_i (and similarly b_i, c_i) and n pairs of values $(m_j(a_i), \beta_{a_j}^i), j = 1, \ldots, n$, except those $(m_j(a_i), \beta_{a_j}^i)$ where both P_i and P_j are corrupt. Finally for each honest party P_i the environment provides values $\Delta_{a_i}^j$ for $j = 1, \ldots, n$.
2. The functionality chooses a, b at random and sets $c = ab$. Now it creates the representation $[a]$ (and similarly for $[b]$ and $[c]$) as follows:
   (a) First it chooses random shares for the honest parties such that the sum of these and those specified by the environment is correct: Let $\mathtt{C}$ be the set of corrupt parties, then a_i is chosen at random for $P_i \notin \mathtt{C}$, subject to $a = \sum_i a_i$.
   (b) For each honest P_i, and $j = 1, \ldots, n$, $\beta_{a_j}^i$ is chosen as follows: if P_j is honest, $\beta_{a_j}^i$ is chosen at random, otherwise it sets $\beta_{a_j}^i = m_i(a_j) - \alpha_j^i a_j$. Note that the environment already specified $m_i(a_j), a_j$, so what is done here is to construct the key to be held by P_i to be consistent with the share and MAC chosen by the environment.
   (c) For all $i = 1, \ldots, n, j = 1, \ldots, n$ it sets $K_{a_j}^i = (\alpha_j^i, \beta_{a_j}^i)$, and computes $m_j(a_i) = \mathsf{MAC}_{K_{a_i}^j}(a_i) + \Delta_{a_i}^j$.
   (d) Now all data for $[a]$ is created. The functionality sends $\{a_i, \{K_{a_j}^i, m_j(a_i)\}_{j=1,\ldots,n}\}$ to each honest P_i (no need to send anything to corrupt parties, the environment already has the data).

Fig. 14. The ideal functionality for making triples $[a], [b], [c]$.