Between a Rock and a Hard Place: Interpolating Between MPC and FHE

A. Choudhury, J. Loftus, E. Orsini, A. Patra and N.P. Smart

Dept. Computer Science,
University of Bristol,
United Kingdom.
{Ashish.Choudhary,Emmanuela.Orsini,Arpita.Patra}@bristol.ac.uk,
{loftus,nigel}@cs.bris.ac.uk

Abstract. We present a computationally secure MPC protocol for threshold adversaries which is parametrized by a value L. When $L = 2$ we obtain a classical form of MPC protocol in which interaction is required for multiplications, as L increases interaction is reduced in that one requires interaction only after computing a higher degree function. When L approaches infinity one obtains the FHE based protocol of Gentry, which requires no interaction. Thus one can trade communication for computation in a simple way.
Our protocol is based on an interactive protocol for "bootstrapping" a somewhat homomorphic encryption scheme. The key contribution is that our presented protocol is highly communication efficient enabling us to obtain reduced communication when compared to traditional MPC protocols for relatively small values of L.

1 Introduction

In the last few years computing on encrypting data via either Fully Homomorphic Encryption (FHE) or Multi-Party Computation (MPC) has been subject to a remarkable number of improvements. Firstly, FHE was shown to be possible [29]; and this was quickly followed by a variety of applications and performance improvements [8, 11, 10, 26, 27, 37, 38]. Secondly, whilst MPC has been around for over thirty years, only in the last few years we have seen an increased emphasis on practical instantiations; with some very impressive results [7, 16, 35].

We focus on MPC where n parties wish to compute a function on their respective inputs. Whilst the computational overhead of MPC protocols, compared to computing "in the clear", is relatively small (for example in practical protocols such as [23, 35] a small constant multiple of the "in the clear" cost), the main restriction on practical deployment of MPC is the communication cost. Even for protocols in the pre-processing model, the communication cost in terms of number of bits per multiplication gate and per party is a constant multiple of the bit length $\log p$ of the data being manipulated for a typically large value of the constant. This is a major drawback of MPC protocols since communication is generally more expensive than computation. Theoretical results like [21] (for the computational case) and [20] (for the information theoretic case) brings down the per gate per party communication cost to a very small quantity; essentially $\mathcal{O}(\frac{\log n}{n} \cdot \log |C| \cdot \log p)$ bits for a circuit C of size $|C|$. While these results suggest that the communication cost can be asymptotically brought down to a constant for large n, the constants are known to be large for any practical purpose. Our interest lies in constructing efficient MPC protocols where the efficiency is measured in terms of exact complexity rather than the asymptotic complexity.

In his thesis, Gentry [28] showed how FHE can be used to reduce the communication cost of MPC down to virtually zero for any number of parties. In Gentry's MPC protocol all parties encrypt to each other, their input under a shared FHE public key. They then compute the function homomorphically, and at the end perform a shared decryption. This implies an MPC protocol whose communication is limited to a function of the input and output sizes, and not to the complexity of the circuit. However, this reduction in communication complexity comes at a cost, namely the huge expense of evaluating homomorphically the function. With current understanding of FHE technology, this solution is completely infeasible in practice.

A variant of Gentry's protocol was presented in [1] where the parties outsource their computation to a server and only interact via a distributed decryption. The key innovation in [1] was that independently generated (FHE) keys can be combined into a "global" FHE key with distributed decryption capability. We do not assume such a functionality of the keys (but one can easily extend our results to accommodate this); instead we focus on using distributed decryption to enable *efficient* multi-party bootstrapping.

In [23], following on the work in [6], the authors propose an MPC protocol which uses a Somewhat Homomorphic Encryption (SHE) scheme as an "optimization". Based in the preprocessing model, the authors utilize an SHE scheme which can evaluate circuits of multiplicative depth one to optimize the preprocessing step of an essentially standard MPC protocol. The optimizations, and use of SHE, in [23] are focused on the case of computational improvements. In this work we invert the use of SHE in [23], by using the SHE in the online phase of the MPC protocol, so as to optimize the communication efficiency of MPC for any number of parties.

In particular we interpolate between the two extremes of traditional MPC protocols (with high communication but low computational costs) and Gentry's FHE based solution (with high computation but low communication costs). Our interpolation is dependent on a parameter, which we label as L, where $L \geq 2$. At one extreme, for $L = 2$ our protocol resembles traditional MPC protocols, whilst at the other extreme, for $L = \infty$ our protocol is exactly that of Gentry's FHE based solution.

The solution we present is in the pre-processing model; in which we allow a pre-processing phase which can compute data which is neither input nor function dependent. This pre-processed data is consumed in the online phase. As usual in such a model our goal is for efficiency in the online phase only. We present our basic protocol and efficiency analysis for the case of passive threshold adversaries only; i.e. we can tolerate up to t passive corruptions where $t < n$. We then note that security against t active adversaries with $t < n/3$ can be achieved for no extra cost in the online phase. For the active security case, essentially the same communication costs can be achieved even when $t < n/2$, bar some extra work (which is independent of $|C|$) to eliminate the cheating parties when they are detected. The security of our protocols are proven in the UC framework [12] (a brief discussion on the UC framework can be found in Appendix D).

Overview: Our protocol is intuitively simple. We first take an L-levelled SHE scheme (strictly it has $L + 1$ levels, but can evaluate circuits with L levels of multiplications) which possesses a distributed decryption protocol for the specific access structure required by our MPC protocol. We assume that the SHE scheme is implemented over a ring which supports N embeddings of the underlying finite field $\mathbb{F}_p$ into the message space of the SHE scheme. Almost all known SHE schemes support such packing of the finite field into the plaintext slots in an SIMD manner [26, 38]; and such packing has been crucial in the implementation of SHE in various applications [22, 23, 27].

Clearly with such a setup we can implement Gentry's MPC solution for circuits of multiplicative depth L. All that remains is how to "bootstrap" from circuits with multiplicative depth L to arbitrary circuits. The standard solution would be to bootstrap the FHE scheme directly, following the blueprint outlined in Gentry's thesis. However, in the case of applications to MPC we could instead utilize a protocol to perform the bootstrapping. In a nutshell that is exactly what we propose.

The main issue then is show how to efficiently perform the bootstrapping in a distributed manner; where efficiency is measured in terms of computational and communication performance. Naively performing an MPC protocol to execute the bootstrapping phase will lead to a large communication overhead, due to the inherent overhead in dealing with homomorphic encryptions. But on it's own this is enough to obtain our asymptotic interpolation between FHE and MPC; we however aim to provide an efficient and practical interpolation. That is one which is efficient for small values of L. It turns out that a special case of a suitable bootstrapping protocol can be found as a sub-procedure of the MPC protocol in [23]. We extract the required protocol, generalise it, and then apply it to our MPC situation.

To ease exposition we will not utilize the packing from [26] to perform such evaluations; we see this as a computational optimization which is orthogonal to the issues we will explore in this paper. In any practical instantiation of the protocol of this paper such a packing could be used, as described in [26], in evaluating the multiplicative circuit of depth L. However, we will use this packing to perform the bootstrapping in a communication efficient manner.

The bootstrapping protocol runs in two phases. In the first (offline) phase we repeatedly generate sets of ciphertexts, one set for each party, such that all parties learn the ciphertexts but only the given party learns their underlying messages (which are assumed to be packed). The offline phase can be run in either a passive, covert or active security model, irrespective of the underlying access structure of the MPC protocol following ideas from [16]. In the second (online) phase the data to be bootstrapped is packed together, a random mask is added (computed from the offline phase data), a distributed decryption protocol is executed to obtain the masked data which is then re-encrypted, the mask is subtracted and then the data is unpacked. All these steps are relatively efficient, with communication only being required for the distributed decryption.

To apply our interactive bootstrapping method efficiently we need to make a mild assumption on the circuit being evaluated; this is similar to the assumptions used in [20, 21, 24]. The assumption can be intuitively seen as saying that the circuit is relatively wide enough to enable packing of enough values which need to be bootstrapped at each respective level. We expect that most circuits in practice will satisfy our assumption, and we will call the circuits which satisfy our requirement "well formed".

We pause to note that the ability to open data within the MPC protocol enables one to perform more than a simple evaluation of an arithmetic circuit. This observation is well known in the MPC community, where it has been used to obtain efficient protocols for higher level functions [13, 19]. Thus enabling a distributed bootstrapping also enables one to produce more efficient protocols than a purely FHE based one.

We instantiate our protocol with the BGV scheme [9] and obtain sufficient parameter sizes following the methodology in [27, 16]. Due to the way we utilize the BGV scheme we need to restrict to MPC protocols defined over arithmetic circuits over a finite field $\mathbb{F}_p$, with $p \equiv 1 \pmod{m}$ with $m = 2 \cdot N$ and $N = 2^r$ for some r.

We show that even for a very small value of L, in particular $L = 5$, we can achieve better communication efficiency than many practical MPC protocols in the preprocessing model. Most practical MPC protocols such as [7, 23, 35] require the transmission of at least two finite field elements per multiplication gate between each pair of parties. In [23] a technique is presented which can reduce this to the transmission of an average of three field elements per multiplication gate per party (and not per pair of parties). Note the models in [7] (three party, one passive adversary) and [23, 35] (n party, dishonest majority, active security) are different from ours (we assume honest majority, active security); but even mapping these protocols to our setting of n party honest majority would result in the same communication characteristics. We show that for relatively small values of L, i.e. $L > 8$, one can obtain a communication efficiency of less than one field element per gate and party (details available in Section 6).

Clearly, by setting L appropriately one can obtain a communication efficiency which improves upon that in [20, 21]; albeit we are only interested in communication in the online phase of a protocol in the preprocessing model whilst [20, 21] discuss total communication cost over all phases. But we stress this is not in itself interesting, as Gentry's FHE based protocol can beat the communication efficiency of [20, 21] in any case. What is interesting is that we can beat the communication efficiency of the online phase of practical MPC protocols, with very small values of L indeed. Thus the protocol in this paper may provide a practical tradeoff between existing MPC protocols (which consume high bandwidth) and FHE based protocols (which require huge computation).

Our protocol therefore enables the following use-case: it is known that SHE schemes only become prohibitively computationally expensive for large L; indeed one of the reasons why the protocols in [23, 16] are so efficient is that they restrict to evaluating homomorphically circuits of multiplicative depth one. With our protocol parties can a priori decide the value of L, for a value which enables them to produce a computationally efficient SHE scheme. Then they can execute an MPC protocol with communication costs reduced by effectively a factor of L. Over time as SHE technology improves the value of L can be increased and we can obtain Gentry's original protocol. Thus our methodology enables us to interpolate between the case of standard MPC and the eventual goal of MPC with almost zero communication costs.

2 Well Formed Circuits

In this section we define what we mean by well formed circuits, and the pre-processing which we require on our circuits. We take as given an arithmetic circuit C defined over a finite field $\mathbb{F}_p$. In particular the circuit C is a directed acyclic graph consisting of edges made up of n_I input wires, n_O output wires, and n_W internal wires, plus a set of nodes being given by a set of gates $\mathbb{G}$. The gates are divided into sets of Add gates $\mathbb{G}_A$ and Mult gates $\mathbb{G}_M$, with $\mathbb{G} = \mathbb{G}_A \cup \mathbb{G}_M$, with each Add/Mult gate taking two wires (or a constant value in $\mathbb{F}_p$) as input and producing one wire as output. The circuit is such that all input wires are open on their input ends, and all output wires are open on their output ends, with the internal wires being connected on both ends. We let the depth of the circuit d be the length of the maximum path from an input wire to an output wire. Our definition of a well formed circuit is parametrized by two positive integer values N and L.

We now associate inductively to each wire in the circuit an integer valued label as follows. The input wires are given the label one; then all other wires are given a label according to the following rule (where we assume a constant input to a gate has label L)

$$\begin{aligned}\text{Label of output wire of } \mathsf{Add} \text{ gate } &= \min(\text{Label of input wires}),\\ \text{Label of output wire of } \mathsf{Mult} \text{ gate } &= \min(\text{Label of input wires}) - 1.\end{aligned}$$

Thus the minimum value of a label is $1 - d$ (which is negative for a general d). Looking ahead, the reason for starting with an input label of one is when we match this up with our MPC protocol this will result in low communication complexity for the input stage of the computation.

We now augment the circuit, to produce a new circuit C^{aug} which will have labels in the range $[1, \ldots, L]$, by adding in some special gates which we will call Refresh gates; the set of such gates are denoted as $\mathbb{G}_R$. A Refresh gate takes as input a maximum of N wires, and produces as output an exact copy of the specified input wires. The input requirement is that the input wires must have label in the range $[1, \ldots, L]$, and all that the Refresh gate does is relabel the labels of the gate's input wires to be L. At the end of the augmentation process we require the invariant that all wire labels in C^{aug} are then in the range $[1, \ldots, L]$, and the circuit is now essentially a collection of "sub-circuits" of multiplicative depth at most $L - 1$ glued together using Refresh gates. However, we require that this is done with as small a number of Refresh gates as possible.

Definition 1 (Well Formed Circuit). *A circuit C will be called well formed if the number of* Refresh *gates in the associated augmented circuit C^{aug} is atmost*

$$\frac{2 \cdot |\mathbb{G}_M|}{L \cdot N}.$$

We expect that "most" circuits will be well formed due to the following argument: We first note that the only gates which concern us are multiplication gates; so without loss of generality we consider a circuit C consisting only of multiplication gates. The circuit has d layers, and let the width of C (i.e. the number of gates) at layer i be w_i. Consider the algorithm to produce C^{aug} which considers each layer in turn, from $i = 1$ to d and adds $\mathsf{Refresh}$ gates where needed. When reaching level i in our algorithm to produce C^{aug} we can therefore assume (by induction) that all input wires at this layer have labels in the range $[1, \ldots, L]$. To maintain the invariant we only need to apply a $\mathsf{Refresh}$ operation to those input wires which have label one. Let p_i denote the proportion of wires at layer i which have label one when we perform this process. It is clear that the number of required $\mathsf{Refresh}$ gates which we will add into C^{aug} at level i will be at most $\lceil 2 \cdot p_i \cdot w_i / N \rceil$, where the factor of two comes from the fact that each multiplication gate has two input wires.

Assuming a large enough circuit we can assume for most layers that this proportion p_i will be approximately $1/L$. So summing up over all levels, the expected number of $\mathsf{Refresh}$ gates in C^{aug} will be:

$$\sum_{i=1}^{d} \left\lceil \frac{2 \cdot w_i}{L \cdot N} \right\rceil \approx \frac{2}{L \cdot N} \cdot \sum_{i=1}^{d} w_i = \frac{2 \cdot |\mathbb{G}_M|}{L \cdot N}.$$

Note, we would expect that for most circuits this upper bound on the number of $\mathsf{Refresh}$ gates could be easily met. For example our above rough analysis did not take into account the presence of gates with fan-out greater than one (meaning there are less wires to $\mathsf{Refresh}$ than we estimated above), nor did it take into account utilizing unused slots in the $\mathsf{Refresh}$ gates to refresh wires with labels not equal to one.

Determining an optimum algorithm for moving from C to a suitable C^{aug}, with a minimal number of $\mathsf{Refresh}$ gates, is an interesting optimization problem which we leave as an open problem; however clearly the above outlined greedy algorithm will work for most circuits.

3 Threshold L-Levelled Packed Somewhat Homomorphic Encryption (SHE)

Here we summarize the properties we need from our (abstract) underlying SHE scheme. In Appendix A we shall formally expand on the following high level intuition. The scheme will be parametrized by a number of values; namely the security parameter κ, the number of levels L and the amount of packing of plaintext elements which can be made into one ciphertext N. In practice the parameter N will be a function of the number of levels L and the security parameter κ. The message space of the SHE scheme is defined to be $\mathcal{M} = \mathbb{F}_p^N$, and we embed the finite field $\mathbb{F}_p$ into $\mathcal{M}$ via a map $\chi : \mathbb{F}_p \longrightarrow \mathcal{M}$. See Appendix B for a discussion as to the various choices one has for χ when we specialise to the BGV SHE scheme.

Ciphertexts $\mathfrak{c}$ in our scheme will have associated to them a level $\mathfrak{l}$, so in what follows we shall describe ciphertexts as a pair $(\mathfrak{c}, \mathfrak{l})$ which defines the ciphertext and the associated level $\mathfrak{l} \in [0, \ldots, L]$ which it is at. The SHE scheme supports the following usual operations:

1. $\mathsf{SHE.KeyGen}(1^\kappa)$: it produces a public/private key pair $(\mathsf{pk}, \mathsf{sk})$, and an evaluation key ek.
2. $\mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{m}, r)$: it takes a plaintext $\mathbf{m}$ and randomness r, and produces a ciphertext $(\mathfrak{c}, L)$ at level L.
3. $\mathsf{SHE.Dec}_{\mathsf{sk}}(\mathfrak{c}, \mathfrak{l})$: it decrypts $\mathfrak{c}$ to obtain a plaintext $\mathbf{m}$. We say that $\mathbf{m}$ is the *plaintext associated with* $\mathfrak{c}$.
4. $\mathsf{SHE.Eval}_{\mathsf{ek}}(C, (\mathfrak{c}_1, \mathfrak{l}_1^{in}), \ldots, (\mathfrak{c}_{\ell_{in}}, \mathfrak{l}_{\ell_{in}}^{in}))$: This function homomorphically evaluates the arithmetic circuit C, which is a circuit of multiplicative depth at most L, in an SIMD manner producing the output $(\hat{\mathfrak{c}}_1, \mathfrak{l}_1^{out}), \ldots, (\hat{\mathfrak{c}}_{\ell_{out}}, \mathfrak{l}_{\ell_{out}}^{out})$. The procedure $\mathsf{SHE.Eval}$ is obtained by evaluating the arithmetic circuit C via the use of the sub-procedures:
   - $\mathsf{SHE.Add}_{\mathsf{ek}}((\mathfrak{c}_1, \mathfrak{l}_1), (\mathfrak{c}_2, \mathfrak{l}_2)) \to (\mathfrak{c}_{\mathsf{Add}}, \min(\mathfrak{l}_1, \mathfrak{l}_2))$.
   - $\mathsf{SHE.Mult}_{\mathsf{ek}}((\mathfrak{c}_1, \mathfrak{l}_1), (\mathfrak{c}_2, \mathfrak{l}_2)) \to (\mathfrak{c}_{\mathsf{Mult}}, \min(\mathfrak{l}_1, \mathfrak{l}_2) - 1)$.

– $\mathsf{SHE.ScalarMult_{ek}}((\mathfrak{c}_1, \mathfrak{l}_1), \mathbf{a}) \rightarrow (\mathfrak{c}_{\mathsf{Scalar}}, \mathfrak{l}_1)$.

In addition we require the following three operations:

1. $\mathsf{SHE.Pack_{ek}}((\mathfrak{c}_1, \mathfrak{l}_1), \ldots, (\mathfrak{c}_N, \mathfrak{l}_N))$: If $\mathfrak{c}_i$ is a ciphertext with associated plaintext $\chi(m_i)$ then this procedure produces a ciphertext $(\mathfrak{c}, \min(\mathfrak{l}_1, \ldots, \mathfrak{l}_N))$ with associated plaintext $\mathbf{m} = (m_1, \ldots, m_N)$.
2. $\mathsf{SHE.Unpack_{ek}}(\mathfrak{c}, \mathfrak{l})$: If $\mathfrak{c}$ is a ciphertext with associated plaintext $\mathbf{m} = (m_1, \ldots, m_N)$ then this procedure produces N ciphertexts $(\mathfrak{c}_1, \mathfrak{l}), \ldots, (\mathfrak{c}_N, \mathfrak{l})$ such that $\mathfrak{c}_i$ has associated plaintext $\chi(m_i)$.
3. $\mathsf{SHE.LowerLevel_{ek}}((\mathfrak{c}, \mathfrak{l}), \mathfrak{l}')$: This procedure, for $\mathfrak{l}' < \mathfrak{l}$, produces a ciphertext with the same associated plaintext as of $(\mathfrak{c}, \mathfrak{l})$ but at level $\mathfrak{l}'$.

To support threshold decryption we need to modify the key generation algorithm and present additional algorithms for our scheme. The key difference is that the key generation algorithm takes (n, t) as additional input (which defines the threshold properties of our scheme), along with the statistical security parameter sec (for the security of the distributed decryption) and outputs a set of keys $(\mathsf{dk}_1, \ldots, \mathsf{dk}_n)$ for the distributed decryption, one for each party, in addition to the public key, secret key and the evaluation key. The keys for the distributed decryption satisfy the property that given t (or less) such keys no information about the secret key sk can be obtained, while any set of $t + 1$ or more of such keys are sufficient to recover the secret key in the information theoretic sense.

This allows us to define the following three additional algorithms:

1. $\mathsf{SHE.ShareDec_{dk_\mathit{i}}}(\mathfrak{c}, \mathfrak{l})$: Given a ciphertext $\mathfrak{c}$ and a key dk_i for the distributed decryption, this algorithm computes a decryption share $\bar{\mu}_i$ of $\mathfrak{c}$.
2. $\mathsf{SHE.ShareCombine}((\mathfrak{c}, \mathfrak{l}), \{\bar{\mu}_i\}_{i \in [1,\ldots,n]})$: Given a ciphertext $\mathfrak{c}$ and any set of $t + 1$ or more (correct) decryption shares $\bar{\mu}_i$ corresponding to $\mathfrak{c}$, anyone can combine them (using this algorithm) to obtain a plaintext $\mathbf{m}$. We require that the output $\mathbf{m}$ of $\mathsf{SHE.ShareCombine}$ should be the same as would have been obtained by applying the decryption algorithm $\mathsf{SHE.Dec_{sk}}$ on $\mathfrak{c}$.
3. $\mathsf{SHE.ShareSim}((\mathfrak{c}, \mathfrak{l}), \mathbf{m}, I, \{\bar{\mu}_i\}_{i \in I})$: We also require that given a ciphertext $\mathfrak{c}$ (with associated plaintext $\mathbf{m}$) and any decryption share $\bar{\mu}_i$ corresponding to $\mathfrak{c}$ computed using dk_i, no information is revealed about dk_i. This is formalized by requiring that there exists a PPT algorithm $\mathsf{SHE.ShareSim}$, which on input a set of decryption shares from the parties in set I with $|I| \leq t$, can simulate the remaining $(n - t)$ decryption shares $\{\bar{\mu}_j^*\}$, such that the simulated shares when combined with the shares of I, outputs $\mathbf{m}$. Moreover, we want the statistical distance between the simulated and real shares to be at most $2^{-\mathsf{sec}}$.

Full details of the associated syntax, and the associated correctness and security definitions of our threshold L-level packed SHE scheme are given in Appendix A. In Appendix B we instantiate the abstract syntax with a threshold SHE scheme based on the BGV scheme [9]. We pause to note the difference between our underlying SHE scheme, which is just an SHE scheme which supports distributed decryption, and that of [1] which requires a special FHE scheme which is *key homomorphic*.

4 MPC from SHE – The Semi-honest Settings

In this section we present our generic MPC protocol for the computation of any arbitrary depth d circuit using an abstract threshold L-levelled SHE scheme. For the ease of exposition we first concentrate on the case of semi-honest security, and then we deal with active security in Section 5.

Without loss of generality we make the simplifying assumption that the function f to be computed takes a single input from each party and has a single output; specifically $f : \mathbb{F}_p^n \rightarrow \mathbb{F}_p$. The ideal functionality $\mathcal{F}_f$ presented in Figure 1 computes such a given function f, represented by a well formed circuit C.

Functionality $\mathcal{F}_f$

$\mathcal{F}_f$ interacts with the parties $P_1, \ldots, P_n$ and the adversary $\mathcal{S}$ and is parametrized by an n-input function $f : \mathbb{F}_p^n \rightarrow \mathbb{F}_p$.

- Upon receiving (sid, i, x_i) from the party P_i for every $i \in [1, \ldots, n]$ where $x_i \in \mathbb{F}_p$, compute $y = C(x_1, \ldots, x_n)$, send (sid, y) to all the parties and the adversary $\mathcal{S}$ and halt. Here C denotes the (publicly known) well formed arithmetic circuit over $\mathbb{F}_p$ representing the function f.

Fig. 1. The Ideal Functionality for Computing a Given Function

We will present a protocol to realize the ideal functionality $\mathcal{F}_f$ in a hybrid model in which we are given access to an ideal functionality $\mathcal{F}_{\text{SETUPGEN}}$ which implements a distributed key generation for the underlying SHE scheme. In particular the $\mathcal{F}_{\text{SETUPGEN}}$ functionality presented in Figure 2 computes the public key, secret key, evaluation key and the keys for the distributed decryption of an L-levelled SHE scheme, distributes the public key and the evaluation key to all the parties and sends the ith key dk_i (for the distributed decryption) to the party P_i for each $i \in [1, \ldots, n]$. In addition, the functionality also computes a random encryption $\mathfrak{c}_{\mathbf{1}}$ with associated plaintext $\mathbf{1} = (1, \ldots, 1) \in \mathcal{M}$ and sends it to all the parties. Looking ahead, $\mathfrak{c}_{\mathbf{1}}$ will be required while proving the security of our MPC protocol. The ciphertext $\mathfrak{c}_{\mathbf{1}}$ is at level one, as we only need it to pre-multiply the ciphertexts which are going to be decrypted via the distributed decryption protocol; thus the output of a multiplication by $\mathfrak{c}_{\mathbf{1}}$ need only be at level zero. Looking ahead, this ensures that (with respect to our instantiation of SHE) the noise is kept to a minimum at this stage of the protocol.

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

$\mathcal{F}_{\text{SETUPGEN}}$ interacts with the parties $P_1, \ldots, P_n$ and the adversary $\mathcal{S}$ and is parameterized by an L-levelled SHE scheme

- Upon receiving (sid, i) from the party P_i for every $i \in [1, \ldots, n]$, compute $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}, \mathsf{dk}_1, \ldots, \mathsf{dk}_n) = \mathsf{SHE.KeyGen}(1^{\kappa}, 1^{\mathsf{sec}}, n, t)$ and $(\mathfrak{c}_{\mathbf{1}}, 1) = \mathsf{SHE.LowerLevel}_{\mathsf{ek}}((\mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{1}, r), 1)$ for $\mathbf{1} = (1, \ldots, 1) \in \mathcal{M}$ and some randomness r. Finally send $(\mathsf{sid}, \mathsf{pk}, \mathsf{ek}, \mathsf{dk}_i, (\mathfrak{c}_{\mathbf{1}}, 1))$ to the party P_i for every $i \in [1, \ldots, n]$ and halt.

Fig. 2. The Ideal Functionality for Key Generation

4.1 The MPC Protocol in the $\mathcal{F}_{\text{SETUPGEN}}$-hybrid Model

Here we present our MPC protocol Π_f^{SH} in the $\mathcal{F}_{\text{SETUPGEN}}$-hybrid model. Let C be the (well formed) arithmetic circuit representing the function f and C^{aug} be the associated augmented circuit (which includes the necessary Refresh gates). The protocol Π_f^{SH} (see Figure 3) runs in two phases: offline and online. The computation performed in the offline phase is completely independent of the circuit and (private) inputs of the parties and therefore can be carried out well ahead of the time (namely the online phase) when the function and inputs are known. If the parties have more than one input/output then one can apply packing/unpacking at the input/output stages of the protocol; we leave this minor modification to the reader.

In the offline phase, the parties interact with $\mathcal{F}_{\text{SETUPGEN}}$ to obtain the public key, evaluation key and their respective keys for performing distributed decryption, corresponding to a threshold L-levelled SHE scheme.

Protocol Π_f^{SH}

Let C^{aug} denote an augmented circuit for a well formed circuit C over $\mathbb{F}_p$ representing f and let SHE be a threshold L-levelled SHE. Moreover, let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of n parties For the session ID sid the parties do the following:

Offline Computation: Every party $P_i \in \mathcal{P}$ does the following:

– Call $\mathcal{F}_{\text{SETUPGEN}}$ with (sid, i) and receive $(\mathsf{sid}, \mathsf{pk}, \mathsf{ek}, \mathsf{dk}_i, (\mathfrak{c}_{\mathbf{1}}, 1))$.
– Randomly select ζ plaintexts $\mathbf{m}_{i,1}, \ldots, \mathbf{m}_{i,\zeta} \in \mathcal{M}$, compute level L encryptions of these plaintexts $(\mathfrak{c}_{\mathbf{m}_{i,k}}, L) = \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{m}_{i,k}, r_{i,k})$. Send $(\mathsf{sid}, i, (\mathfrak{c}_{\mathbf{m}_{i,1}}, L), \ldots, (\mathfrak{c}_{\mathbf{m}_{i,\zeta}}, L))$ to every party in $\mathcal{P}$.
– Upon receiving $(\mathsf{sid}, j, (\mathfrak{c}_{\mathbf{m}_{j,1}}, L), \ldots, (\mathfrak{c}_{\mathbf{m}_{j,\zeta}}, L))$ from all parties $P_j \in \mathcal{P}$, apply $\mathsf{SHE.Add}$ for $1 \le k \le \zeta$, on $(\mathfrak{c}_{\mathbf{m}_{1,k}}, L), \ldots, (\mathfrak{c}_{\mathbf{m}_{n,k}}, L)$, set the resultant ciphertext as the kth offline ciphertext $\mathfrak{c}_{\mathbf{m}_k}$ with the (unknown) associated plaintext $\mathbf{m}_k = \mathbf{m}_{1,k} + \cdots + \mathbf{m}_{n,k}$.

Online Computation: Every party $P_i \in \mathcal{P}$ does the following:

– **Input Stage**: On having input $x_i \in \mathbb{F}_p$, compute $(\mathfrak{c}_{\mathbf{x}_i}, 1) = \mathsf{SHE.LowerLevel}_{\mathsf{ek}}(\mathsf{SHE.Enc}_{\mathsf{pk}}(\chi(x_i), r_i), 1)$ with randomness r_i and send $(\mathsf{sid}, i, (\mathfrak{c}_{\mathbf{x}_i}, 1))$ to each party. Receive $(\mathsf{sid}, j, (\mathfrak{c}_{\mathbf{x}_j}, 1))$ from each party $P_j \in \mathcal{P}$.
– **Computation Stage**: Associate the ciphertexts received in the previous stage with the corresponding input wires of C^{aug} and then homomorphically evaluate the circuit C^{aug} gate by gate as follows:
  - **Addition Gate and Multiplication Gate**: Given $(\mathfrak{c}_1, \mathfrak{l}_1)$ and $(\mathfrak{c}_2, \mathfrak{l}_2)$ associated with the input wires of the gate where $\mathfrak{l}_1, \mathfrak{l}_2 \in [1, \ldots, L]$, locally compute $(\mathfrak{c}, \mathfrak{l}) = \mathsf{SHE.Add}_{\mathsf{ek}}((\mathfrak{c}_1, \mathfrak{l}_1), (\mathfrak{c}_2, \mathfrak{l}_2))$ with $\mathfrak{l} = \min(\mathfrak{l}_1, \mathfrak{l}_2)$ for an addition gate and $(\mathfrak{c}, \mathfrak{l}) = \mathsf{SHE.Mult}_{\mathsf{ek}}((\mathfrak{c}_1, \mathfrak{l}_1), (\mathfrak{c}_2, \mathfrak{l}_2))$ with $\mathfrak{l} = \min(\mathfrak{l}_1, \mathfrak{l}_2) - 1$ for a multiplication gate. Associate $(\mathfrak{c}, \mathfrak{l})$ with the output wire of the gate.
  - **Refresh Gate**: For the kth refresh gate in the circuit, the kth offline ciphertext $(\mathfrak{c}_{\mathbf{m}_k}, L)$ is used. Let $(\mathfrak{c}_1, \mathfrak{l}_1), \ldots, (\mathfrak{c}_N, \mathfrak{l}_N)$ be the ciphertexts associated with the input wires of the refresh gate where $\mathfrak{l}_1, \ldots, \mathfrak{l}_N \in [1, \ldots, L]$:
    * **Packing**: Locally compute $(\mathfrak{c}_{\mathbf{z}}, \mathfrak{l}) = \mathsf{SHE.Pack}_{\mathsf{ek}}(\{(\mathfrak{c}_i, \mathfrak{l}_i)\}_{i \in [1,\ldots,N]})$ where $\mathfrak{l} = \min(\mathfrak{l}_1, \ldots, \mathfrak{l}_N)$.
    * **Masking**: Locally compute $(\mathfrak{c}_{\mathbf{z}+\mathbf{m}_k}, 0) = \mathsf{SHE.Add}_{\mathsf{ek}}(\mathsf{SHE.Mult}_{\mathsf{ek}}((\mathfrak{c}_{\mathbf{z}}, \mathfrak{l}), (\mathfrak{c}_{\mathbf{1}}, 1)), (\mathfrak{c}_{\mathbf{m}_k}, L))$
    * **Decrypting**: Locally compute the decryption share $\bar{\mu}_i = \mathsf{SHE.ShareDec}_{\mathsf{dk}_i}(\mathfrak{c}_{\mathbf{z}+\mathbf{m}}, 0)$ and send $(\mathsf{sid}, i, \bar{\mu}_i)$ to every other party. On receiving $(\mathsf{sid}, j, \bar{\mu}_j)$ from every $P_j \in \mathcal{P}$, compute the plaintext $\mathbf{z} + \mathbf{m}_k = \mathsf{SHE.ShareCombine}((\mathfrak{c}_{\mathbf{z}+\mathbf{m}_k}, 0), \{\bar{\mu}_j\}_{j \in [1,\ldots,n]})$.
    * **Re-encrypting**: Locally re-encrypt $\mathbf{z} + \mathbf{m}_k$ by computing $(\hat{\mathfrak{c}}_{\mathbf{z}+\mathbf{m}_k}, L) = \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{z} + \mathbf{m}_k, r)$ using a publicly known (common) randomness r.
    * **Unmasking**: Locally subtract $(\mathfrak{c}_{\mathbf{m}_k}, L)$ from $(\hat{\mathfrak{c}}_{\mathbf{z}+\mathbf{m}_k}, L)$ to obtain $(\hat{\mathfrak{c}}_{\mathbf{z}}, L)$.
    * **Unpacking**: Locally compute $(\hat{\mathfrak{c}}_1, L), \ldots, (\hat{\mathfrak{c}}_N, L) = \mathsf{SHE.Unpack}_{\mathsf{ek}}(\hat{\mathfrak{c}}_{\mathbf{z}}, L)$ and associate $(\hat{\mathfrak{c}}_1, L), \ldots, (\hat{\mathfrak{c}}_N, L)$ with the output wires of the refresh gate.
– **Output Stage**: Let $(\mathfrak{c}, \mathfrak{l})$ be the ciphertext associated with the output wire of C^{aug} where $\mathfrak{l} \in [1, \ldots, L]$.
  - **Randomization:** Compute a random encryption $(\mathfrak{c}_i, L) = \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{0}, r_i')$ of $\mathbf{0} = (0, \ldots, 0)$ and send $(\mathsf{sid}, i, (\mathfrak{c}_i, L))$ to every other party. On receiving $(\mathsf{sid}, j, (\mathfrak{c}_j, L))$ from every $P_j \in \mathcal{P}$, apply $\mathsf{SHE.Add}$ on $\{(\mathfrak{c}_j, L)\}_{j \in [1,\ldots,n]}$ to obtain $(\mathfrak{c}_{\mathbf{0}}, L)$. Compute $(\hat{\mathfrak{c}}, 0) = \mathsf{SHE.Add}_{\mathsf{ek}}(\mathsf{SHE.Mult}_{\mathsf{ek}}((\mathfrak{c}, \mathfrak{l}), (\mathfrak{c}_{\mathbf{1}}, 1)), (\mathfrak{c}_{\mathbf{0}}, L))$.
  - **Output Decryption:** Compute the decryption share $\bar{\gamma}_i = \mathsf{SHE.ShareDec}_{\mathsf{dk}_i}(\hat{\mathfrak{c}}, 0)$ and send $(\mathsf{sid}, i, \bar{\gamma}_i)$ to every party. On receiving $(\mathsf{sid}, j, \bar{\gamma}_j)$ from every $P_j \in \mathcal{P}$, compute $\mathbf{y} = \mathsf{SHE.ShareCombine}((\hat{\mathfrak{c}}, 0), \{\bar{\gamma}_j\}_{j \in [1,\ldots,n]})$, output y and halt, where $y = \chi^{-1}(\mathbf{y})$.

Fig. 3. The Protocol for Realizing $\mathcal{F}_f$ against a Semi-Honest Adversary in the $\mathcal{F}_{\text{SETUPGEN}}$-hybrid Model

Next each party sends encryptions of ζ random elements and then additively combines them to generate ζ ciphertexts at level L of truly random elements (unknown to the adversary). Here ζ is assumed to be large enough, so that for a typical circuit it is more than the number of refresh gates in the circuit, i.e. $\zeta > \mathbb{G}_R$. Looking ahead, these random ciphertexts created in the offline phase are used in the online phase to evaluate refresh gates by (homomorphically) masking the messages associated with the input wires of a refresh gate.

During the online phase, the parties encrypt their private inputs using the public key and distribute the corresponding ciphertexts to all other parties. These ciphertexts are transmitted at level one, thus consuming low bandwidth, and are then elevated to level L by the use of a following $\mathsf{Refresh}$ gate (which would have been inserted by the circuit augmentation process). Note that the inputs of the parties are in $\mathbb{F}_p$ and so the parties first apply the mapping χ (embedding $\mathbb{F}_p$ into the message space $\mathcal{M}$ of SHE) before encrypting their private inputs.

The input stage is followed by the homomorphic evaluation of C^{aug} as follows: The addition and multiplication gates are evaluated locally using the addition and multiplication algorithm of the SHE. For each refresh gate, the parties execute the following protocol to enable a "multiparty bootstrapping" of the input ciphertexts: the parties pick one of the random ciphertext created in the offline phase (for each refresh gate a different ciphertext is used) and perform the following computation to refresh N ciphertexts with levels in the range $[1, \ldots, L]$ and obtain N fresh level L ciphertexts, with the associated messages unperturbed:

- Let $(\mathfrak{c}_1, \mathfrak{l}_1) \ldots, (\mathfrak{c}_N, \mathfrak{l}_N)$ be the N ciphertexts with associated plaintexts $\chi(z_1), \ldots, \chi(z_N)$ with every $z_i \in \mathbb{F}_p$, that need to be refreshed (i.e. they are the inputs of a refresh gate).
- The N ciphertexts are then (locally) packed into a single ciphertext $\mathfrak{c}$, which is then homomorphically masked with a random ciphertext from the offline phase.
- The resulting masked ciphertext is then publicly opened via distributed decryption, This allows for the creation of a fresh encryption of the opened value at level L.
- The resulting fresh encryption is then homomorphically unmasked so that its associated plaintext is the same as original plaintext prior to the original masking.
- This fresh (unmasked) ciphertext is then unpacked to obtain N fresh ciphertexts, having the same associated plaintexts as the original N ciphertexts $\mathfrak{c}_i$ but at level L.

By packing the ciphertexts together we only need to invoke distributed decryption once, instead of N times. This leads to a more communication efficient online phase, since the distributed decryption is the only operation that demands communication. Without affecting the correctness of the above technique, but to ensure security, we add an additional step while doing the masking: the parties homomorphically pre-multiply the ciphertext $\mathfrak{c}$ with $\mathfrak{c}_{\mathbf{1}}$ before masking. Recall that $\mathfrak{c}_{\mathbf{1}}$ is an encryption of $\mathbf{1} \in \mathcal{M}$ generated by $\mathcal{F}_{\text{SETUPGEN}}$ and so by doing the above operation, the plaintext associated with $\mathfrak{c}$ remains the same. During the simulation in the security proof, this step allows the simulator to set the decrypted value to the random mask (irrespective of the circuit inputs), by playing the role of $\mathcal{F}_{\text{SETUPGEN}}$ and replacing $\mathfrak{c}_{\mathbf{1}}$ with $\mathfrak{c}_{\mathbf{0}}$, a random encryption of $\mathbf{0} = (0, \ldots, 0)$. Furthermore, this step explains the reason why we made provision for an extra multiplication during circuit augmentation by insisting that the refresh gates take inputs with labels in $[1, \ldots, L]$, instead of $[0, \ldots, L]$; the details are available in the simulation proof of security of our MPC protocol.

Finally, the function output y is obtained by another distributed decryption of the output ciphertext. However, this step is also not secure unless the ciphertext is randomized again by pre-multiplication by $\mathfrak{c}_{\mathbf{1}}$ and adding n encryptions of $\mathbf{0}$ where each party contributes one encryption. In the simulation, the simulator gives encryption of $\chi(y)$ on behalf of one honest party and replaces $\mathfrak{c}_{\mathbf{1}}$ by $\mathfrak{c}_{\mathbf{0}}$, letting the output ciphertext correspond to the actual output y, even though the circuit is evaluated with zero as the inputs of the honest parties during the simulation (the simulator will not know the real inputs of the honest parties and thus will simulate them with zero). A similar idea was also used in [17]; details can be found in the security proof.

Intuitively, privacy follows because at any stage of the computation, the keys of the honest parties for the distributed decryption are not revealed and so the adversary will not be able to decrypt any intermediate ciphertext. Correctness follows from the properties of the SHE and the fact that the level of each ciphertext in the protocol remains in the range $[1, \ldots, L]$, thanks to the refresh gates. So even though the circuit C may have any arbitrary depth $d > L$, we can homomorphically evaluate C using an L-levelled SHE.

Theorem 1. *Let $f : \mathbb{F}_p^n \to \mathbb{F}_p$ be a function over $\mathbb{F}_p$ represented by a well formed arithmetic circuit C of depth d over $\mathbb{F}_p$. Let $\mathcal{F}_f$ (presented in Figure 1) be the ideal functionality computing f and let* SHE *be a threshold L-levelled SHE scheme. Then the protocol Π_f^{SH} UC-secure realizes $\mathcal{F}_f$ against a static, semi-honest adversary $\mathcal{A}$, corrupting upto $t < n$ parties in the $\mathcal{F}_{\mathrm{SETUPGEN}}$-hybrid Model.*

The proof of this theorem can be found in Appendix E.

5 MPC from SHE – The Active Setting

The functionalities from Section 4 are in the passive corruption model. In the presence of an active adversary, the functionalities will be modified as follows: the respective functionality considers the input received from the majority of the parties and performs the task it is supposed to do on those inputs. For example, in the case of $\mathcal{F}_f$, the functionality considers for the computation those x_is, corresponding to the P_is from which the functionality has received the message (sid, i, x_i); on the behalf of the remaining P_is, the functionality substitutes 0 as the default input for the computation. Similarly for $\mathcal{F}_{\mathrm{SETUPGEN}}$, the functionality performs its task if it receives the message (sid, i) from the majority of the parties. These are the standard notions of defining ideal functionalities for various corruption scenarios and we refer [30] for the complete formal details; we will not present separately the ideal functionality $\mathcal{F}_f$ and $\mathcal{F}_{\mathrm{SETUPGEN}}$ for the malicious setting.

A closer look at Π_f^{SH} shows that we can "compile" it into an actively secure MPC protocol tolerating t active corruptions if we ensure that every corrupted party "proves" in a zero knowledge (ZK) fashion that it constructed the following correctly: **(1)** The ciphertexts in the offline phase; **(2)** The ciphertexts during the input stage and **(3)** The randomizing ciphertexts during the output stage.

Apart from the above three requirements, we would also require a "robust" $\mathsf{SHE.ShareCombine}$ which works correctly even if up to t input decryption shares are incorrect. In Appendix F we show that for our specific SHE scheme, the $\mathsf{SHE.ShareCombine}$ algorithm (based on the standard error-correction) is indeed robust, provided $t < n/3$. For the case of $t < n/2$ we show (in Appendix F) that by including additional steps and incorporating additional zero-knowledge proofs (namely proof of correct decryption), one can also obtain a robust output. Interestingly the MPC protocol requires the transmission of at most $\mathcal{O}(n^3)$ such additional zero-knowledge proofs; i.e. the communication needed to obtain robustness is independent of the circuit being computed. We stress that $t < n/2$ is the *optimal resilience* for computationally secure MPC against active corruptions (with fairness) [31]. In this paper, to keep the protocol presentation and its proof simple, we assume a robust $\mathsf{SHE.ShareCombine}$ (i.e. for the case of $t < n/3$), which applies error correction for the correct decryption.

Before presenting the actively secure MPC protocol, we present in Figure 4 an ideal ZK functionality $\mathcal{F}_{\mathsf{ZK}}^{R}$, parametrized with an NP-relation R, which will be used in our protocol. We next apply this functionality to the following relations to obtain the functionalities $\mathcal{F}_{\mathsf{ZK}}^{R_{enc}}$ and $\mathcal{F}_{\mathsf{ZK}}^{R_{zeroenc}}$.

- $R_{enc} = \{((\mathfrak{c}, \mathfrak{l}), (\mathbf{x}, r)) \mid (\mathfrak{c}, \mathfrak{l}) = \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{x}, r) \text{ if } \mathfrak{l} = L \vee (\mathfrak{c}, \mathfrak{l}) = \mathsf{SHE.LowerLevel}_{\mathsf{ek}}(\mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{x}, r)) \text{ if } \mathfrak{l} = 1\}$: we require this relation to hold for the offline stage ciphertexts (where $\mathfrak{l} = L$) and for the input stage ciphertexts (where $\mathfrak{l} = 1$).
- $R_{zeroenc} = \{((\mathfrak{c}, L), (\mathbf{x}, r)) \mid (\mathfrak{c}, L) = \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{x}, r) \wedge \mathbf{x} = \mathbf{0}\}$: we require this relation to hold for the randomizing ciphertexts during the output stage.

We are now ready to present the protocol Π_f^{MAL} (see Figure 5) in the $(\mathcal{F}_{\mathrm{SETUPGEN}}, \mathcal{F}_{\mathsf{ZK}}^{R_{enc}}, \mathcal{F}_{\mathsf{ZK}}^{R_{zeroenc}})$-hybrid model and assuming a robust $\mathsf{SHE.ShareCombine}$ based on error-correction (i.e. for the case $t < n/3$).

Functionality $\mathcal{F}_{\mathrm{ZK}}^{R}$

$\mathcal{F}_{\mathrm{ZK}}^{R}$ interacts with a prover $P_i \in \{P_1, \ldots, P_n\}$ and the set of n verifiers $\mathcal{P} = \{P_1, \ldots, P_n\}$ and the adversary $\mathcal{S}$.

- Upon receiving $(\mathsf{sid}, i, (x, w))$ from the prover $P_i \in \{P_1, \ldots, P_n\}$, the functionality sends (sid, i, x) to all the verifiers in $\mathcal{P}$ and $\mathcal{S}$ if $R(x, w)$ is true. Else it sends $(\mathsf{sid}, i, \perp)$ and halts.

Fig. 4. The Ideal Functionality for ZK

Protocol Π_f^{MAL}

Let C be the well formed arithmetic circuit over $\mathbb{F}_p$ representing the function f, let C^{aug} denote an augmented circuit associated with C, and let SHE be a threshold L-levelled SHE scheme. For session ID sid the parties in $\mathcal{P} = \{P_1, \ldots, P_n\}$ do the following:

Offline Computation: Every party $P_i \in \mathcal{P}$ does the following:

- Call $\mathcal{F}_{\mathrm{SETUPGEN}}$ with (sid, i) and receive $(\mathsf{sid}, \mathsf{pk}, \mathsf{ek}, \mathsf{dk}_i, (\mathfrak{c}_{\mathbf{1}}, 1))$.
- Same as in the offline phase of Π_f^{SH}, except that for every message $\mathbf{m}_{ik}$ for $k \in [1, \ldots, \zeta]$ and the corresponding ciphertext $(\mathfrak{c}_{\mathbf{m}_{ik}}, L) = \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{m}_{ik}, r_{ik})$, call $\mathcal{F}_{\mathrm{ZK}}^{R_{enc}}$ with $(\mathsf{sid}, i, ((\mathfrak{c}_{\mathbf{m}_{ik}}, L), (\mathbf{m}_{ik}, r_{ik})))$. Receive $(\mathsf{sid}, j, (\mathfrak{c}_{\mathbf{m}_{jk}}, L))$ from $\mathcal{F}_{\mathrm{ZK}}^{R_{enc}}$ for $k \in [1, \ldots, \zeta]$ corresponding to each $P_j \in \mathcal{P}$. If $(\mathsf{sid}, j, \perp)$ is received from $\mathcal{F}_{\mathrm{ZK}}^{R_{enc}}$ for some $P_j \in \mathcal{P}$, then consider ζ publicly known level L encryptions of random values from $\mathcal{M}$ as $(\mathfrak{c}_{\mathbf{m}_{jk}}, L)$ for $k \in [1, \ldots, \zeta]$.

Online Computation: Every party $P_i \in \mathcal{P}$ does the following:

- **Input Stage**: On having input $x_i \in \mathbb{F}_p$, compute level L ciphertext $(\mathfrak{c}_{\mathbf{x}_i}, 1) = \mathsf{SHE.LowerLevel}_{\mathsf{ek}}(\mathsf{SHE.Enc}_{\mathsf{pk}}(\chi(x_i), r_i), 1)$ with randomness r_i and call $\mathcal{F}_{\mathrm{ZK}}^{R_{enc}}$ with the message $(\mathsf{sid}, i, ((\mathfrak{c}_{\mathbf{x}_i}, 1), (\chi(x_i), r_i)))$. Receive $(\mathsf{sid}, j, (\mathfrak{c}_{\mathbf{x}_j}, 1))$ from $\mathcal{F}_{\mathrm{ZK}}^{R_{enc}}$ corresponding to each $P_j \in \mathcal{P}$. If $(\mathsf{sid}, j, \perp)$ is received from $\mathcal{F}_{\mathrm{ZK}}^{R_{enc}}$ for some $P_j \in \mathcal{P}$, then consider a publicly known level 1 encryption of $\chi(0)$ as $(\mathfrak{c}_{\mathbf{x}_j}, 1)$ for such a P_j.
- **Computation Stage**: Same as Π_f^{SH}, except that now the robust $\mathsf{SHE.ShareCombine}$ is used.
- **Output Stage**: Let $(\mathfrak{c}, \mathfrak{l})$ be the ciphertext associated with the output wire of C^{aug} where $\mathfrak{l} \in [1, \ldots, L]$.
  - **Randomization:** Compute a random encryption $(c_i, L) = \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{0}, r_i')$ of $\mathbf{0} = (0, \ldots, 0)$ and call $\mathcal{F}_{\mathrm{ZK}}^{R_{zeroenc}}$ with the message $(\mathsf{sid}, i, ((c_i, L), (\mathbf{0}, r_i')))$. Receive $(\mathsf{sid}, j, (c_j, L))$ from $\mathcal{F}_{\mathrm{ZK}}^{R_{zeroenc}}$ corresponding to each $P_j \in \mathcal{P}$. If $(\mathsf{sid}, j, \perp)$ is received from $\mathcal{F}_{\mathrm{ZK}}^{R_{zeroenc}}$ for some $P_j \in \mathcal{P}$, then consider a publicly known level L encryption of $\mathbf{0}$ as (c_j, L) for such a P_j.
  - The rest of the steps are same as in Π_f^{SH}, except that now the robust $\mathsf{SHE.ShareCombine}$ is used.

Fig. 5. The Protocol for Realizing $\mathcal{F}_f$ against an Active Adversary in the $(\mathcal{F}_{\mathrm{SETUPGEN}}, \mathcal{F}_{\mathrm{ZK}}^{R_{enc}}, \mathcal{F}_{\mathrm{ZK}}^{R_{zeroenc}})$-hybrid Model

Theorem 2. *Let $f : \mathbb{F}_p^n \rightarrow \mathbb{F}_p$ be a function represented by a well-formed arithmetic circuit C over $\mathbb{F}_p$. Let $\mathcal{F}_f$ (presented in Figure 1) be the ideal functionality computing f and let* SHE *be a threshold L-levelled SHE scheme such that* $\mathsf{SHE.ShareCombine}$ *is robust. Then the protocol Π_f^{MAL} UC-secure realizes $\mathcal{F}_f$ in the $(\mathcal{F}_{\mathrm{SETUPGEN}}, \mathcal{F}_{\mathrm{ZK}}^{R_{enc}}, \mathcal{F}_{\mathrm{ZK}}^{R_{zeroenc}})$-hybrid Model against a static, active adversary $\mathcal{A}$ corrupting t parties.*

See appendix F for the proof of the theorem. In the same appendix, we further present a more efficient offline phase attaining a linear communication overhead (asymptotically) in the number of preprocessed ciphertexts. We note that UC-secure realizations of $\mathcal{F}_{\mathrm{ZK}}^{R_{enc}}$ and $\mathcal{F}_{\mathrm{ZK}}^{R_{zeroenc}}$ can be obtained in the CRS model using similar techniques as used in [2]. Finally we do not worry about the instantiation of $\mathcal{F}_{\mathrm{SETUPGEN}}$ as we consider it a one time set-up, which can be done via standard techniques (such as running an MPC protocol).

6 Estimating the Consumed Bandwidth

In Appendix C we determine the parameters for the instantiation of our SHE scheme using BGV by adapting the analysis from [27] and [16]. In this section we use this parameter estimation to show that our MPC protocol can infact give improved communication complexity compared to the standard MPC protocols, for relatively small values of the parameter L. We are interested in the communication cost of our online stage computation. To ease our exposition we will focus on the passively secure case from section 4; the analysis for the active security case with $t < n/3$ is exactly the same (bar the additional cost of the exchange of zero-knowledge proofs for the input stage and the output stage). For the case of active security with $t < n/2$ we also need to add in the communication related to the dispute control strategy outlined in Appendix F for attaining robust $\mathsf{SHE.ShareCombine}$ with $t < n/2$; but this is a cost which is proportional to $\mathcal{O}(n^3)$.

To get a feel for the parameters from Appendix C, we now specialise to the case of finite fields of size $p \approx 2^{64}$, statistical security parameter sec of 40, and for various values of the computational security level κ. Resolving the various inequalities (from Appendix C), we then estimate in Table 1 the value of N, assuming a small value for n (we need to restrict to small n to ensure a large enough range in the PRF needed in the distributed decryption protocol; see Appendix B.4).

L	$\kappa = 80$	$\kappa = 128$	$\kappa = 256$
2	16384	16384	32768
3	16384	16384	32768
4	16384	32768	32768
5	32768	32768	65536
6	32768	32768	65536
7	32768	32768	65536
8	32768	65536	65536
9	32768	65536	65536
10	65536	65536	65536

Table 1. The value of N for various values of κ and L

Since a $\mathsf{Refresh}$ gate requires the transmission of $n - 1$ elements (namely the decryption shares) in the ring R_{q_0} from party P_i to the other parties, the total communication in our protocol (in bits) is

$$|\mathbb{G}_R| \cdot n \cdot (n-1) \cdot |R_{q_0}|,$$

where $|R_{q_0}|$ is the number of bits needed to transmit an element in R_{q_0}, i.e. $N \cdot \log_2 p_0$. Assuming the circuit meets our requirement of being well formed, this implies that total communication cost for our protocol is

$$\frac{2 \cdot |\mathbb{G}_M| \cdot n \cdot (n-1) \cdot N \cdot \log_2 p_0}{L \cdot N} = \frac{2 \cdot n \cdot (n-1) \cdot |\mathbb{G}_M|}{L} \cdot \log_2(309 \cdot 2^{\mathsf{sec}} \cdot p \cdot \sqrt{N}).$$

Using the batch distributed technique (of efficiently and parallely evaluating $t+1$ independent $\mathsf{Refresh}$ gates simultaneously) from Appendix B.6 this can be reduced to

$$\mathsf{Cost} = \frac{4 \cdot n \cdot (n-1) \cdot |\mathbb{G}_M|}{L \cdot (t+1)} \cdot \log_2(309 \cdot 2^{\mathsf{sec}} \cdot p \cdot \sqrt{N}).$$

We are interested in the *overhead per multiplication gate*, in terms of equivalent numbers of finite field elements in $\mathbb{F}_p$, which is given by $\mathsf{Cost}/(|\mathbb{G}_M| \cdot \log_2 p)$, and the cost per party is $\mathsf{Cost}/(|\mathbb{G}_M| \cdot n \cdot \log_2 p)$.

At the 128 bit security level, with $p \approx 2^{64}$, and $\mathsf{sec} = 40$ (along with the above estimated values of N), this means for $n = 3$ parties, and at most $t = 1$ corruption, we obtain the following cost estimates:

L		2	3	4	5	6	7	8	9	10
Total Cost	$\mathsf{Cost}/(\vert\mathbb{G}_M\vert \cdot \log_2 p)$	12.49	8.33	6.31	5.05	4.21	3.61	3.19	2.84	2.55
Per party Cost	$\mathsf{Cost}/(\vert\mathbb{G}_M\vert \cdot n \cdot \log_2 p)$	4.16	2.77	2.10	1.68	1.40	1.20	1.06	0.94	0.85

Note for $L = 2$ our protocol becomes the one which requires interaction after every multiplication, for $L = 3$ we require interaction only after every two multiplications and so on. Note that most practical MPC protocols in the preprocessing model have a per gate per party communication cost of at least 2 finite field elements, e.g. [23]. Thus, even when $L = 5$, we obtain better communication efficiency in the online phase than traditional practical protocols in the preprocessing model with these parameters.

7 Acknowledgements

This work has been supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO, by EPSRC via grant EP/I03126X, and by Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under agreement number FA8750-11-2-0079. The second author was supported by an Trend Micro Ltd, and the fifth author was supported by in part 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 Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.

References

1. G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2012.
2. G. Asharov, A. Jain, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. *IACR Cryptology ePrint Archive*, 2011:613, 2011.
3. Z. BeerliováTrubíniová and M. Hirt. Perfectly-Secure MPC with Linear Communication Complexity. In R. Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer Verlag, 2008.
4. Z. BeerliováTrubíniová and Martin Hirt. Efficient Multi-party Computation with Dispute Control. In S. Halevi and T. Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 305–328. Springer Verlag, 2006.
5. R. Bendlin and I. Damgård. Threshold Decryption and Zero-Knowledge Proofs for Lattice-Based Cryptosystems. In D. Micciancio, editor, *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*, pages 201–218. Springer Verlag, 2010.
6. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188, 2011.
7. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2008.
8. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.
9. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325. ACM, 2012.
10. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106. IEEE, 2011.

11. Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.
12. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
13. Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *Financial Cryptography*, volume 6052 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2010.
14. T. M. Cover and J. A. Thomas. *Elements of information theory*. Wiley, 2006.
15. R. Cramer, I. Damgård, and Y. Ishai. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In J. Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 342–362. Springer Verlag, 2005.
16. I. Damgard, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical Covertly Secure MPC for Dishonest Majority – or: Breaking the SPDZ Limits. Cryptology ePrint Archive, Report 2012/642, 2012. `http://eprint.iacr.org/`.
17. I. Damgård and J. B. Nielsen. Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264. Springer Verlag, 2003.
18. I. Damgård and J. B. Nielsen. Scalable and Unconditionally Secure Multiparty Computation. In A. Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer Verlag, 2007.
19. Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multiparty computation for equality, comparison, bits and exponentiation. In *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006.
20. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2010.
21. Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 241–261, 2008.
22. Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. Implementing AES via an actively/covertly secure dishonest-majority mpc protocol. In *SCN*, volume 7485 of *Lecture Notes in Computer Science*, pages 241–263. Springer, 2012.
23. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.
24. Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation for boolean circuits in the preprocessing model. *IACR Cryptology ePrint Archive*, 2012:512, 2012.
25. Matthias Fitzi and Martin Hirt. Optimally Efficient Multi-valued Byzantine Agreement. In Eric Ruppert and Dahlia Malkhi, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 163–168. ACM, 2006.
26. C. Gentry, S. Halevi, and N. P. Smart. Fully Homomorphic Encryption with Polylog Overhead. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer Verlag, 2012.
27. C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.
28. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.
29. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009.
30. O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
31. Martin Hirt and Jesper Buus Nielsen. Robust Multiparty Computation with Linear Communication Complexity. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 463–482. Springer Verlag, 2006.
32. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011.
33. Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. Cloud-assisted multiparty computation from fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2011:663, 2011.
34. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.

35. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2012.
36. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.
37. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.
38. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Appeared in Designs, Codes and Cryptography*, 2012.

A Threshold L-Levelled Packed Somewhat Homomorphic Encryption

In this Appendix we present a detailed explanation of the syntax and requirements for our Threshold L-Levelled Packed Somewhat Homomorphic Encryption Scheme.

Let $\mathcal{C}(L)$ denote the family of circuits consisting of addition and multiplication gates whose labels follow the conventions in Section 2; except that input wires have label L and whose minimum wire label is zero. Thus $\mathcal{C}(L)$ is the family of standard arithmetic circuits of multiplicative depth at most L which consist of 2-input addition and multiplication gates over $\mathbb{F}_p$, whose wire labels lie in the range $[0, \ldots, L]$. Informally, an L-levelled SHE scheme supports homomorphic evaluation of any circuit in the family $\mathcal{C}(L)$ where the input wire values v_i are mapped to ciphertexts (at level L) by encrypting $\chi(v_i)$.

As remarked in the introduction we could also, as in [26], extend the circuit family $\mathcal{C}(L)$ to include gates which process N input values at once as

$$N\text{-}\mathsf{Add}\,(\langle u_1, \ldots, u_N\rangle, \langle v_1, \ldots, v_N\rangle) := \langle u_1 + v_1, \ldots, u_N + v_N\rangle,$$
$$N\text{-}\mathsf{Mult}\,(\langle u_1, \ldots, u_N\rangle, \langle v_1, \ldots, v_N\rangle) := \langle u_1 \times v_1, \ldots, u_N \times v_N\rangle.$$

But such an optimization of the underlying circuit is orthogonal to our consideration. However, the underlying L-levelled packed SHE scheme supports such operations on its underlying plaintext (we will just not consider these operations in our circuits being evaluated).

We can evaluate however subcircuits in $\mathcal{C}(L)$; and this is how we will describe the homomorphic evaluation below (this will later help us to argue the correctness property of our general MPC protocol). In particular if $C \in \mathcal{C}(L)$, we can deal with sub-circuits C^{sub} of C whose input wires have labels $\mathfrak{l}_1^{in}, \ldots, \mathfrak{l}_{\ell_{in}}^{in}$, and whose output wires have labels $\mathfrak{l}_1^{out}, \ldots, \mathfrak{l}_{\ell_{out}}^{out}$, where $\mathfrak{l}_i^{in}, \mathfrak{l}_i^{out} \in [0, \ldots, L]$. Then given ciphertexts $\mathfrak{c}_1, \ldots, \mathfrak{c}_{\ell_{in}}$ encrypting the messages $\mathbf{m}_1, \ldots, \mathbf{m}_{\ell_{in}}$, for which the ciphertexts are at level $\mathfrak{l}_1^{in}, \ldots, \mathfrak{l}_{\ell_{in}}^{in}$, the homomorphic evaluation function will produce ciphertexts $\hat{\mathfrak{c}}_1, \ldots, \hat{\mathfrak{c}}_{\ell_{out}}$, at levels $\mathfrak{l}_1^{out}, \ldots, \mathfrak{l}_{\ell_{out}}^{out}$, which encrypt the messages corresponding to evaluating C^{sub} on the components of the vectors $\mathbf{m}_1, \ldots, \mathbf{m}_{\ell_{in}}$ in an SIMD manner. More formally:

Definition 2 (Threshold L-levelled Packed SHE). *An L-levelled public key packed somewhat homomorphic encryption (SHE) scheme with the underlying message space $\mathcal{M} = \mathbb{F}_p^N$, public key space $\mathcal{PK}$, secret key space $\mathcal{SK}$, evaluation key space $\mathcal{EK}$, ciphertext space $\mathcal{CT}$ and distributed decryption key space $\mathcal{DK}_i$ for $i \in [1, \ldots, n]$ is a collection of the following PPT algorithms, parametrized by a computational security parameter κ and a statistical security parameter* sec*:*

1. $\mathsf{SHE.KeyGen}(1^{\kappa}, 1^{\mathsf{sec}}, n, t) \to (\mathsf{pk}, \mathsf{ek}, \mathsf{sk}, \mathsf{dk}_1, \ldots, \mathsf{dk}_n)$*: The key generation algorithm outputs a public key* $\mathsf{pk} \in \mathcal{PK}$*, a public evaluation key* $\mathsf{ek} \in \mathcal{EK}$*, a secret key* $\mathsf{sk} \in \mathcal{SK}$ *and n keys* $(\mathsf{dk}_1, \ldots, \mathsf{dk}_n)$ *for the distributed decryption, with* $\mathsf{dk}_i \in \mathcal{DK}_i$*.*

2. $\mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{m}, r) \rightarrow (\mathfrak{c}, L)$*: The encryption algorithm computes a ciphertext* $\mathfrak{c} \in \mathcal{CT}$*, which encrypts a plaintext vector* $\mathbf{m} \in \mathcal{M}$ *under the public key* pk *using the randomness*[1] r *and outputs* $(\mathfrak{c}, L)$ *to indicate that the* associated level *of the ciphertext is* L*.*
3. $\mathsf{SHE.Dec}_{\mathsf{sk}}(\mathfrak{c}, \mathfrak{l}) \rightarrow \mathbf{m}'$*: The decryption algorithm decrypts a ciphertext* $\mathfrak{c} \in \mathcal{CT}$ *of associated level* $\mathfrak{l}$ *where* $\mathfrak{l} \in [0, \ldots, L]$ *using the decryption key* sk *and outputs a plaintext* $\mathbf{m}' \in \mathcal{M}$*. We say that* $\mathbf{m}'$ *is the* plaintext associated with $\mathfrak{c}$.
4. $\mathsf{SHE.ShareDec}_{\mathsf{dk}_i}(\mathfrak{c}, \mathfrak{l}) \rightarrow \bar{\mu}_i$*: The share decryption algorithm takes as input a ciphertext* $\mathfrak{c}$ *with associated level* $\mathfrak{l} \in [0, \ldots, L]$*, a key* dk_i *for the distributed decryption, and computes a decryption share* $\bar{\mu}_i$ *of* $\mathfrak{c}$*.*
5. $\mathsf{SHE.ShareCombine}((\mathfrak{c}, \mathfrak{l}), \{\bar{\mu}_i\}_{i \in [1,\ldots,n]}) \rightarrow \mathbf{m}'$*: The share combine algorithm takes as input a ciphertext* $\mathfrak{c}$ *with associated level* $\mathfrak{l} \in [0, \ldots, L]$ *and a set of* n *decryption shares and outputs a plaintext* $\mathbf{m}' \in \mathcal{M}$*.*
6. $\mathsf{SHE.Eval}_{\mathsf{ek}}(C^{\mathsf{sub}}, (\mathfrak{c}_1, \mathfrak{l}_1^{in}), \ldots, (\mathfrak{c}_{\ell_{in}}, \mathfrak{l}_{\ell_{in}}^{in})) \rightarrow (\hat{\mathfrak{c}}_1, \mathfrak{l}_1^{out}), \ldots, (\hat{\mathfrak{c}}_{\ell_{out}}, \mathfrak{l}_{\ell_{out}}^{out})$*: The homomorphic evaluation algorithm is a deterministic polynomial time algorithm (polynomial in* L, ℓ_{in}, ℓ_{out} *and* κ*) that takes as input the evaluation key* ek*, a sub-circuit* C^{sub} *of a circuit* $C \in \mathcal{C}(L)$ *with* ℓ_{in} *input gates and* ℓ_{out} *output gates as well as a set of* ℓ_{in} *ciphertexts* $\mathfrak{c}_1, \ldots, \mathfrak{c}_{\ell_{in}}$*, with associated level* $\mathfrak{l}_1^{in}, \ldots, \mathfrak{l}_{\ell_{in}}^{in}$*, and outputs* ℓ_{out} *ciphertexts* $\hat{\mathfrak{c}}_1, \ldots, \hat{\mathfrak{c}}_{\ell_{out}}$*, with associated levels* $\mathfrak{l}_1^{out}, \ldots, \mathfrak{l}_{\ell_{out}}^{out}$ *respectively, where each* $\mathfrak{l}_i^{in}, \mathfrak{l}_i^{out} \in [0, \ldots, L]$ *is the label associated to the given input/output wire in* C^{sub}*.*
   Algorithm SHE.Eval *associates the input ciphertexts with the input gates of* C^{sub} *and homomorphically evaluates* C^{sub} *gate by gate in an SIMD manner on the components of the input messages. For this,* SHE.Eval *consists of separate algorithms* SHE.Add *and* SHE.Mult *for homomorphically evaluating addition and multiplication gates respectively. More specifically, given two ciphertexts* $(\mathfrak{c}_1, \mathfrak{l}_1)$ *and* $(\mathfrak{c}_2, \mathfrak{l}_2)$ *with associated levels* $\mathfrak{l}_1$ *and* $\mathfrak{l}_2$ *respectively where* $\mathfrak{l}_1, \mathfrak{l}_2 \in [0, \ldots, L]$ *then*[2]*:*
   - $\mathsf{SHE.Add}_{\mathsf{ek}}((\mathfrak{c}_1, \mathfrak{l}_1), (\mathfrak{c}_2, \mathfrak{l}_2)) \rightarrow (\mathfrak{c}_{\mathsf{Add}}, \min(\mathfrak{l}_1, \mathfrak{l}_2))$*: The deterministic polynomial time addition algorithm takes as input* $(\mathfrak{c}_1, \mathfrak{l}_1), (\mathfrak{c}_2, \mathfrak{l}_2)$ *and outputs a ciphertext* $\mathfrak{c}_{\mathsf{Add}}$ *with associated level* $\min(\mathfrak{l}_1, \mathfrak{l}_2)$*.*
   - $\mathsf{SHE.Mult}_{\mathsf{ek}}((\mathfrak{c}_1, \mathfrak{l}_1), (\mathfrak{c}_2, \mathfrak{l}_2)) \rightarrow (\mathfrak{c}_{\mathsf{Mult}}, \min(\mathfrak{l}_1, \mathfrak{l}_2) - 1)$*: The deterministic polynomial time multiplication algorithm takes as input* $(\mathfrak{c}_1, \mathfrak{l}_1), (\mathfrak{c}_2, \mathfrak{l}_2)$ *and outputs a ciphertext* $\mathfrak{c}_{\mathsf{Mult}}$ *with associated level* $\min(\mathfrak{l}_1, \mathfrak{l}_2) - 1$*.*
   - $\mathsf{SHE.ScalarMult}_{\mathsf{ek}}((\mathfrak{c}_1, \mathfrak{l}_1), \mathbf{a}) \rightarrow (\mathfrak{c}_{\mathsf{Scalar}}, \mathfrak{l}_1)$*: The deterministic polynomial time scalar multiplication algorithm takes as input* $(\mathfrak{c}_1, \mathfrak{l}_1)$ *and a plaintext* $\mathbf{a} \in \mathcal{M}$ *and outputs a ciphertext* $\mathfrak{c}_{\mathsf{Scalar}}$ *with associated level* $\mathfrak{l}_1$*.*
7. $\mathsf{SHE.Pack}_{\mathsf{ek}}((\mathfrak{c}_1, \mathfrak{l}_1), \ldots, (\mathfrak{c}_N, \mathfrak{l}_N)) \rightarrow (\mathfrak{c}, \min(\mathfrak{l}_1, \ldots, \mathfrak{l}_N))$*: If* $\mathfrak{c}_i$ *is a ciphertext with associated plaintext* $\chi(m_i)$*, then this procedure produces a ciphertext* $(\mathfrak{c}, \min(\mathfrak{l}_1, \ldots, \mathfrak{l}_N))$ *with associated plaintext* $\mathbf{m} = (m_1, \ldots, m_N)$*.*
8. $\mathsf{SHE.Unpack}_{\mathsf{ek}}(\mathfrak{c}, \mathfrak{l}) \rightarrow ((\mathfrak{c}_1, \mathfrak{l}), \ldots, (\mathfrak{c}_N, \mathfrak{l}))$*: If* $\mathfrak{c}$ *is a ciphertext with associated plaintext* $\mathbf{m} = (m_1, \ldots, m_N)$*, then this procedure produces* N *ciphertexts* $(\mathfrak{c}_1, \mathfrak{l}), \ldots, (\mathfrak{c}_N, \mathfrak{l})$ *such that* $\mathfrak{c}_i$ *has associated plaintext* $\chi(m_i)$*.*
9. $\mathsf{SHE.LowerLevel}_{\mathsf{ek}}((\mathfrak{c}, \mathfrak{l}), \mathfrak{l}') \rightarrow (\mathfrak{c}, \mathfrak{l}')$*: This procedure, for* $\mathfrak{l}' < \mathfrak{l}$*, produces a ciphertext with the same associated plaintext as* $(\mathfrak{c}, \mathfrak{l})$*, but at level* $\mathfrak{l}'$*.*

□

We require the following homomorphic property to be satisfied:

[1] In the paper, unless it is explicitly specified, we assume that some randomness has been used for encryption.

[2] Without loss of generality we assume that we can perform homomorphic operations on ciphertexts of different levels, since we can always deterministically downgrade the ciphertext level of any ciphertext to any value between zero and its current value using $\mathsf{SHE.LowerLevel}_{\mathsf{ek}}$.

– *Somewhat Homomorphic SIMD Property*: Let $C^{\mathsf{sub}} : \mathbb{F}_p^{\ell_{in}} \to \mathbb{F}_p^{\ell_{out}}$ be any sub-circuit of a circuit C in the family $\mathcal{C}(L)$ with respective inputs $\mathbf{m}_1, \ldots, \mathbf{m}_{\ell_{in}} \in \mathcal{M}$, such that C when evaluated N times in an SIMD fashion on the N components of the vectors $\mathbf{m}_1, \ldots, \mathbf{m}_{\ell_{in}}$, produces N sets of ℓ_{out} output values $\hat{\mathbf{m}}_1, \ldots, \hat{\mathbf{m}}_{\ell_{in}} \in \mathcal{M}$. Moreover, for $i \in [1, \ldots, \ell_{in}]$ let $\mathfrak{c}_i$ be a ciphertext of level $\mathfrak{l}_i^{in}$ with associated plaintext vector $\mathbf{m}_i$ and let $(\hat{\mathfrak{c}}_1, \mathfrak{l}_1^{out}), \ldots, (\hat{\mathfrak{c}}_{\ell_{out}}, \mathfrak{l}_{\ell_{out}}^{out}) = \mathsf{SHE.Eval}_{\mathsf{ek}}(C, (\mathfrak{c}_1, \mathfrak{l}_1^{in}), \ldots, (\mathfrak{c}_{\ell_{in}}, \mathfrak{l}_{\ell_{in}}^{in}))$. Then the following holds with probability one for each $i \in [1, \ldots, \ell_{out}]$:

$$\mathsf{SHE.Dec}_{\mathsf{sk}}(\hat{\mathfrak{c}}_i, \mathfrak{l}_i^{out}) = \hat{\mathbf{m}}_i.$$

We also require the following security properties:

– *Key Generation Security*: Let S and D_i be the random variables which denote the probability distribution with which the secret key sk and the ith key dk_i for the distributed decryption is selected from $\mathcal{SK}$ and $\mathcal{DK}_i$ by $\mathsf{SHE.KeyGen}$ for $i = 1, \ldots, n$. Moreover, for a set $I \subseteq \{1, \ldots, n\}$, let D_I denote the random variable which denote the probability distribution with which the set of keys for the distributed decryption, belonging to the indices in I, are selected from the corresponding $\mathcal{DK}_i$s by $\mathsf{SHE.KeyGen}$. Then the following two properties hold:
  - *Correctness*: For any set $I \subseteq \{1, \ldots, n\}$ with $|I| \geq t + 1$, $H(S|D_I) = 0$. Here $H(X|Y)$ denotes the conditional entropy of a random variable X with respect to a random variable Y [14].
  - *Privacy*: For any set $I \subset \{1, \ldots, n\}$ with $|I| \leq t$, $H(S|D_I) = H(S)$.

– *Semantic Security*: For every set $I \subset \{1, \ldots, n\}$ with $|I| \leq t$ and all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following game is negligible in κ:
  - *Key Generation*: The challenger runs $\mathsf{SHE.KeyGen}(1^\kappa, 1^{\mathsf{sec}}, n, t)$ to obtain $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}, \mathsf{dk}_1, \ldots, \mathsf{dk}_n)$ and sends pk, ek and $\{\mathsf{dk}_i\}_{i \in I}$ to $\mathcal{A}$.
  - *Challenge*: $\mathcal{A}$ sends plaintexts $\mathbf{m}_0, \mathbf{m}_1 \in \mathcal{M}$ to the challenger, who randomly selects $b \in \{0, 1\}$ and sends $(\mathfrak{c}, L) = \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{m}_b, r)$ for some randomness r to $\mathcal{A}$.
  - *Output*: $\mathcal{A}$ outputs b'.

  The advantage of $\mathcal{A}$ in the above game is defined to be $|\frac{1}{2} - \Pr[b' = b]|$.

– *Correct Share Decryption*: For any $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}, \mathsf{dk}_1, \ldots, \mathsf{dk}_n)$ obtained as the output of $\mathsf{SHE.KeyGen}$, the following should hold for any ciphertext $(\mathfrak{c}, \mathfrak{l})$ with associated level $\mathfrak{l} \in [0, \ldots, L]$:

$$\mathsf{SHE.Dec}_{\mathsf{sk}}(\mathfrak{c}, \mathfrak{l}) = \mathsf{SHE.ShareCombine}((\mathfrak{c}, \mathfrak{l}), \{\mathsf{SHE.ShareDec}_{\mathsf{dk}_i}(\mathfrak{c}, \mathfrak{l})\}_{i \in [1, \ldots, n]}).$$

– *Share Simulation Indistinguishability*: There exists a PPT simulator $\mathsf{SHE.ShareSim}$, which on input a subset $I \subset \{1, \ldots, n\}$ of size at most t, a ciphertext $(\mathfrak{c}, \mathfrak{l})$ of level $\mathfrak{l} \in [0, \ldots, L]$, a plaintext $\mathbf{m}$ and $|I|$ decryption shares $\{\bar{\mu}_i\}_{i \in I}$ outputs $n - |I|$ simulated decryption shares $\{\bar{\mu}_j^*\}_{j \in \bar{I}}$ with the following property: For any $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}, \mathsf{dk}_1, \ldots, \mathsf{dk}_n)$ obtained as the output of $\mathsf{SHE.KeyGen}$, any subsets $I \subset \{1, \ldots, n\}$ of size at most t, any $\mathbf{m} \in \mathcal{M}$ and any $(\mathfrak{c}, \mathfrak{l})$ where $\mathbf{m} = \mathsf{SHE.Dec}_{\mathsf{sk}}(\mathfrak{c}, \mathfrak{l})$, the following distributions are statistically indistinguishable:

$$(\{\bar{\mu}_i\}_{i \in I}, \mathsf{SHE.ShareSim}((\mathfrak{c}, \mathfrak{l}), \mathbf{m}, \{\bar{\mu}_i\}_{i \in I})) \overset{s}{\approx} \left(\{\bar{\mu}_i\}_{i \in I}, \{\bar{\mu}_j\}_{j \in \bar{I}}\right),$$

  where for all $i \in [1, \ldots, n]$, $\bar{\mu}_i = \mathsf{SHE.ShareDec}_{\mathsf{dk}_i}(\mathfrak{c}, \mathfrak{l})$. We require in particular that the statistical distance between the two distributions is bounded by $2^{-\mathsf{sec}}$. Moreover $\mathsf{SHE.ShareCombine}((\mathfrak{c}, \mathfrak{l}), \{\bar{\mu}_i\}_{i \in I} \cup \mathsf{SHE.ShareSim}((\mathfrak{c}, \mathfrak{l}), \mathbf{m}, \{\bar{\mu}_i\}_{i \in I}))$ outputs the result $\mathbf{m}$. Here $\bar{I}$ denotes the complement of the set I; i.e. $\bar{I} = \{1, \ldots, n\} \setminus I$.

B Instantiating our FHE using BGV

In this section we show an instantiation of SHE based on the scheme of Brakerski, Gentry and Vaikuntanathan (BGV) ([9]). As in [5] we make use of Shamir secret sharing to share the secret key among the parties and pseudorandom secret sharing (PRSS) [15] to non-interactively share a pseudorandom value from a *chosen* interval. We describe a variant of the BGV-type cryptosystems based on the ring learning with error (RLWE) assumption ([34]), naturally supporting the packing operations described in Appendix A.

B.1 Preliminaries

Plaintext Space: We define the polynomial ring $R := \mathbb{Z}[x]/(f(x))$, where $f(x)$ is a monic irreducible polynomial. For our purposes it will suffice to fix $f(x)$ as the cyclotomic polynomial $\Phi_m(x) = x^{m/2} + 1$ with m a power of two. We set $N = \phi(m) = m/2$, where ϕ is the Euler totient function. The ring R is the ring of integers of the mth cyclotomic number field $\mathbb{Q}(\zeta_m)$, with ζ_m an mth root of unity. Denote by $R_q := R/qR$, for an integer q the reduction of R modulo q, i.e. the set of all integer polynomials of degree at most $N-1$ with coefficients in $(-q/2, q/2]$.

Looking ahead the plaintext space of the scheme will be defined to be $R_p := R/pR$ for some prime p such that $p \equiv 1 \mod m$. Since $p \equiv 1 \pmod m$, the polynomial $\Phi_m(x)$ splits into distinct linear factors $F_i(x)$ modulo p:

$$\mathcal{M} := R_p \cong \mathbb{Z}_p[x]/F_1(x) \times \cdots \times \mathbb{Z}_p[x]/F_N(x) \cong \mathbb{F}_p^N,$$

where each factor corresponds to an independent "plaintext slot", holding an element of the finite field $\mathbb{F}_p$. Thus each message $\mathbf{m} \in \mathcal{M}$ actually corresponds to N messages in $\mathbb{F}_p$ and can be represented as an N-vector $(\mathbf{m} \mod F_i)_{i=1,\ldots,N}$. By the Chinese Remainder Theorem addition and multiplication in R_p correspond to SIMD (Single Instruction Multiple Data) operations on the slots and this allows to process N input values at once as described in Appendix Section A.

If we consider the Galois group Gal of $\mathbb{Q}(\zeta_m)$, then $\mathsf{Gal} = \mathsf{Gal}(\mathbb{Q}(\zeta_m)/\mathbb{Q}) \cong \mathbb{Z}_m^*$ and it is formed by the mappings $\sigma_i : a(x) \mapsto a(x^i) \mod \Phi_m(x)$ for all $i \in \mathbb{Z}_m^*$. It is well known ([26]) that Gal transitively acts on plaintext slots, i.e. $\forall i, j \in \{1, \ldots, N\}$ there exists an element $\sigma_{i \to j} \in \mathsf{Gal}$ which sends an element in slot i to an element in slot j.

Random Values: During our construction we will need to sample elements from different distributions over R_q. We will use the following distributions over R, and then map to R_q as appropriate.

– $\mathcal{HWT}(h, N)$: This generates a vector of length N with elements chosen from $\{-1, 0, 1\}$ such that the number of non-zero elements is equal to h.
– $\mathcal{ZO}(0.5, N)$: This generates a vector of length N with elements chosen from $\{-1, 0, 1\}$ such that the coefficient probabilities are $p_{-1} = 1/4$, $p_0 = 1/2$ and $p_1 = 1/4$.
– $\mathcal{DG}(\sigma^2, N)$: This generates a vector of length N with elements chosen according to the discrete Gaussian distribution $D_{\mathbb{Z}^N,\sigma}$.
– $\mathcal{RC}(0.5, \sigma^2, N)$: This generates a triple of elements (a, b, c) where a is sampled from $\mathcal{ZO}_s(0.5, N)$ and b and c are sampled from $\mathcal{DG}_s(\sigma^2, N)$.
– $\mathcal{U}(q, N)$: This generates a vector of length N with elements generated uniformly modulo q.

Pseudorandom Secret Sharing Over Polynomial Rings: Pseudorandom secret sharing was first introduced in [15]. Given a setup, a PRSS scheme allows parties to generate almost unlimited number of Shamir sharings of pseudorandom values at the cost of no communication. Furthermore, the setup is generated once and for all and therefore can be reused many times. While known PRSS works over fields or rings [15, 5], for our purposes we will require a PRSS defined over the polynomial rings $R_{q_{\mathfrak{l}}}$.

In [15] the construction of a PRSS was presented. This was used in [5] to construct a PRSS over $\mathbb{Z}_q$, where $q = \prod p_i$ for n parties, such that each p_i is prime with $p_i > n$. This construction immediately extends to R_q by computing the underlying PRF N times. For completeness we overview the construction here: Given an element $s \in R_q$, we use $[s]$ for the Shamir's sharing of s, $[s]_i = s_i$ for the ith component of the sharing of s, $i = 1, \ldots, n$. We assume a prior one-time setup which distributes a vector of shared keys $\mathbf{k}_A = (\mathsf{k}_{0,A}, \ldots, \mathsf{k}_{N-1,A})$ to each party in A for every subset A of size $n - t$. These keys will be used as the keys of a keyed pseudorandom function PRF family, $\{\psi_{\mathsf{k}}(\cdot)\}_{\mathsf{k} \in \mathbb{K}}$. The pseudorandomness of the output of the following algorithm can be reduced to the PRF security of the underlying PRF at the cost of security loss by a factor of $1/N$.

1. The parties in $\mathcal{P}$ agree on N elements $t_j \in \mathbb{Z}_q$ for $j \in \{0, \ldots, N-1\}$.
2. For $j = 0, \ldots, N-1$, every party $P_i \in \mathcal{P}$ computes $[s_j]_i = \sum_{A \subset \mathcal{P}: |A| = n-t, P_i \in A} \psi_{\mathsf{k}_{j,A}}(t_j) \cdot f_A(i)$. Where $f_A(X)$ denotes the polynomial of degree at most t, such that $f_A(0) = 1$ and $f_A(l) = 0$ for every $P_l \notin A$.
3. For $j = 0, \ldots, N-1$, the value $s_j = \sum_{A \subset \mathcal{P}: |A| = n-t, P_i \in A} \psi_{\mathsf{k}_{j,A}}(t_j)$ denotes the jth pseudorandom shared value from $\mathbb{Z}_q$. Define the associated element in R_q by the polynomial $\sum s_j X^j$.

If the underlying PRF family has range $[-T, \ldots, T]$ over $\mathbb{Z}_q$ then the output of the above PRSS is an element in R_q whose coefficients lie in the range $[-\binom{n}{t}T, \binom{n}{t}T]$. To ease notation we write $\mathbf{s} = \sum_{A \subset \mathcal{P}: |A| = n-t, P_i \in A} \psi_{\mathbf{k}_A}(\mathbf{t})$ for the shared value in R_q, and $[\mathbf{s}]_i = \sum_{A \subset \mathcal{P}: |A| = n-t, P_i \in A} \psi_{\mathbf{k}_A}(\mathbf{t}) \cdot f_A(i)$ for the shares themselves. We note that in general $\binom{n}{t}$ becomes exponentially large, specially if t is a constant fraction of n; however in most practical applications of threshold cryptography, the number of parties n is indeed expected to be small.

Canonical Embedding Norm: Here we recall some results on cyclotomic fields that we need to estimate the parameters of our protocol instantiations. For details regarding properties of canonical norms we refer to [27, 26, 23]. Given a polynomial $a \in R$ we denote by $\|\mathbf{a}\|_\infty = \max_{0 \le i \le N-1} |a_i|$ the standard l_∞-norm. All estimates of noise are taken with respect to the *canonical embedding norm* $\|a\|_\infty^{\mathsf{can}} = \|\sigma(a)\|_\infty$, where σ is the canonical embedding $R \to \mathbb{C}^{\phi(m)}$ defined by $\sigma : a \mapsto a(\zeta_m^k)$, $k \in \mathbb{Z}_m^*$ and ζ_m a fixed primitive mth root of unity. When $a \in R_q$, for some modulus q, we need the canonical embedding norm reduced modulo q:

$$|a|_q^{\mathsf{can}} = \min\{\|a'\|_\infty^{\mathsf{can}} : a' \in R \text{ and } a' \equiv a \pmod{q}\}.$$

To map from norms in the canonical embedding to norms on the coefficients of the polynomials defining the elements in R we note that we have $\|\mathbf{a}\|_\infty \le c_m \cdot \|a\|_\infty^{\mathsf{can}}$, where c_m is the *ring constant*. Since we fix the choice of our base field polynomial as a 2^kth cyclotomic polynomial, we have $c_m = 1$.

B.2 The Basic L-levelled Packed BGV-type Cryptosystem

We review the BGV L-levelled Packed SHE scheme. The scheme is parametrized by a security parameter κ, for a fixed number of levels $L + 1$. Note, we use $L + 1$ levels in our scheme description to make the presentation consistent with the abstract scheme from Appendix A. For $\mathfrak{l} = 0, \ldots, L$, fix a chain of moduli $q_{\mathfrak{l}} = \prod_{i=0}^{\mathfrak{l}} p_i$, with p_i a prime number. Encryption generates level L ciphertexts with respect to the largest modulus q_L. In the $\mathfrak{l}$th level of the scheme ciphertexts consist of two elements in $R_{q_\mathfrak{l}}$, $\mathfrak{l} = 0, \ldots, L$. Throughout homomorphic evaluation we will force a *universal* bound B on the noise contained in ciphertexts (when measured in the canonical embedding norm reduced modulo q) after a $\mathsf{SHE.LowerLevel}$ execution. Since $\|\mathbf{a}\|_\infty \le \|a\|_\infty^{\mathsf{can}} \le B$ this provides an upper bound also on the coefficients used in the underlying decryption algorithm, for such outputs of $\mathsf{SHE.LowerLevel}$. For a description of the algorithm $\mathsf{SHE.LowerLevel}$ see [27]; where it is called *modulus switching*.

However, when applying decryption, or distributed decryption, we will apply the procedure to a ciphertext which is not the direct output of a $\mathsf{SHE.LowerLevel}$ operation. In particular we assume that the canonical norm of the noise of an element passed to the decryption procedure will be bounded by B_{dec}. The decryption procedures will then return the correct output if we have $B_{\mathsf{dec}} \le q_0/2$. For distributed decryption we will need to "boost" this bound to $2^{\mathsf{exp}} \cdot B_{\mathsf{dec}}$, where exp is a "closeness parameter" relating to the statistical security parameter sec. Thus distributed decryption will be work if and only if $2^{\mathsf{exp}} \cdot B_{\mathsf{dec}} < q_0/2$. Below we specify the basic algorithms for the BGV scheme; we will then discuss the extensions to cope with the full syntax of our scheme in Definition 2.

Before presenting the methods we need to pause briefly to remind the reader about modulus switching: A ciphertext at level $\mathfrak{l}$ is given by a pair $\mathfrak{c} = (c_0, c_1) \in R_{q_\mathfrak{l}}^2$ and the decryption procedure computes, for the global secret key $\mathsf{sk} \in R$,

$$[c_0 - \mathsf{sk} \cdot c_1]_{q_\mathfrak{l}} = c_0 - \mathsf{sk} \cdot c_1 \pmod{q_\mathfrak{l}}$$

where we take the symmetric modular operation in the range $[-q_\mathfrak{l}/2, \ldots, q_\mathfrak{l}/2]$. The value $[c_0 - \mathsf{sk} \cdot c_1]_{q_\mathfrak{l}}$ can be interpreted as an element in R, and the associated noise value of the ciphertext is the canonical norm of this element. After each homomorphic operation the norm of the noise in the ciphertexts increases. To reduce it the modulus switching technique ([10, 9]) is used. This procedure takes as input a ciphertext $\mathfrak{c} = (c_0, c_1) \in R_{q_\mathfrak{l}}^2$, with estimated noise ν and transforms it into a ciphertext $\mathfrak{c}' \in R_{q_{\mathfrak{l}'}}^2$ at level $\mathfrak{l}'$, with noise magnitude ν', by scaling down $\mathfrak{c}$ by a factor $q_{\mathfrak{l}'}/q_\mathfrak{l}$ and then rounding to get back an integer ciphertext. The ciphertext $\mathfrak{c}' = (c_0', c_1')$ satisfies $[c_0 - \mathsf{sk} \cdot c_1]_{q_\mathfrak{l}} \equiv [c_0' - \mathsf{sk} \cdot c_1']_{q_{\mathfrak{l}'}} \mod p$ and $\nu' < \nu$. This modulus switching operation corresponds to our operation $\mathsf{SHE.LowerLevel}$ from Definition 2.

1. $\mathsf{SHE.KeyGen}(1^\kappa) \to (\mathsf{pk}, \mathsf{ek}, \mathsf{sk})$: Outputs a secret key $\mathsf{sk} \leftarrow \mathcal{HWT}(h, N)$, a common public key $\mathsf{pk} = (a, b)$ such that $a \leftarrow \mathcal{U}_s(q_L, N)$ and $b = a \cdot \mathsf{sk} + p \cdot e$, with $e \leftarrow \mathcal{DG}(\sigma^2, N)$. This algorithm also outputs the evaluation key ek which consists of $N+1$ public "key-switching matrices" $W_{\mathsf{sk}^2 \to \mathsf{sk}}$ and $W_{\sigma_i(\mathsf{sk}) \to \mathsf{sk}}$ and $\sigma_i \in \mathsf{Gal}$ for $i = 1, \ldots, N$. See [27] for how these are defined.
2. $\mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{m}) \to (\mathfrak{c}, L)$: Given a plaintext $\mathbf{m} \in R_p$, the encryption algorithm samples $(v, e_0, e_1) \leftarrow \mathcal{RC}_s(0.5, \sigma^2, N)$ and then computes in R_{q_L},

$$c_0 = b \cdot v + p \cdot e_0 + \mathbf{m} \quad \text{and} \quad c_1 = a \cdot v + p \cdot e_1.$$

3. $\mathsf{SHE.Dec}_{\mathsf{sk}}(\mathfrak{c}, \mathfrak{l}) \to \mathbf{m}'$: Note, this algorithm is never called in our scheme, we just present it here so as to define correctness and to define what we mean by a message associated to a ciphertext. The algorithm takes as input a ciphertext $\mathfrak{c} = (c_0, c_1) \in R_{q_\mathfrak{l}}^2$ and outputs a plaintext $\mathbf{m}' \in R_p$. This algorithm uses the secret key sk to compute

$$\mu = c_0 - \mathsf{sk} \cdot c_1 = \mathbf{m}' + p \cdot (e \cdot v + e_0 - s \cdot e_1) = \mathbf{m}' + p \cdot u$$

in $R_{q_\mathfrak{l}}$ and then obtains $\mathbf{m}' = (\mu \mod p)$. We denote by ν the estimated noise magnitude obtained by using the canonical embedding norm and we require that $\nu < B_{\mathsf{dec}}$. This decryption procedure will correctly work if $B_{\mathsf{dec}} < q_\mathfrak{l}/2$.

4. $\mathsf{SHE.Eval}_{\mathsf{ek}}(C^{\mathsf{sub}}, (\mathfrak{c}_1, \mathfrak{l}_1^{in}), \ldots, (\mathfrak{c}_{\ell_{in}}, \mathfrak{l}_{\ell_{in}}^{in})) \rightarrow (\hat{\mathfrak{c}}_1, \mathfrak{l}_1^{out}), \ldots, (\hat{\mathfrak{c}}_{\ell_{out}}, \mathfrak{l}_{\ell_{out}}^{out})$: This consists of three separate algorithm SHE.Add, SHE.Mult and SHE.ScalarMult for homomorphically evaluating addition and multiplication gates.
   - $\mathsf{SHE.Add}_{\mathsf{ek}}((\mathfrak{c}_1, \mathfrak{l}_1), (\mathfrak{c}_2, \mathfrak{l}_2))$: It produces a ciphertext $\mathfrak{c}_{\mathsf{Add}}$ in $R_{q_\mathfrak{l}}^2$, with $\mathfrak{l} = \min\{\mathfrak{l}_1, \mathfrak{l}_2\}$. This is performed by first applying $\mathfrak{c}_i' = \mathsf{SHE.LowerLevel}_{\mathsf{ek}}((\mathfrak{c}_i, \mathfrak{l}_i), \mathfrak{l})$ and then taking the coordinate-wise addition of $\mathfrak{c}_1'$ and $\mathfrak{c}_2'$. The noise magnitude of the resulting ciphertext is at most the sum of the noise in $\mathfrak{c}_1$ and $\mathfrak{c}_2$.
   - $\mathsf{SHE.Mult}_{\mathsf{ek}}((\mathfrak{c}_1, \mathfrak{l}_1), (\mathfrak{c}_2, \mathfrak{l}_2))$: This produces a ciphertext $\mathfrak{c}_{\mathsf{Mult}}$ in $R_{q_\mathfrak{l}}^2$, with $\mathfrak{l} = \min\{\mathfrak{l}_1, \mathfrak{l}_2\} - 1$. This is done in one of two ways (so as to minimize the overall parameter sizes in our scheme).
     - If $\mathfrak{l} \neq 1$ then one first applies $\mathfrak{c}_i' = \mathsf{SHE.LowerLevel}_{\mathsf{ek}}((\mathfrak{c}_i, \mathfrak{l}_i), \mathfrak{l})$, then the resulting ciphertexts are tensored. This results in a ciphertext $\tilde{\mathfrak{c}}$ is a vector of higher dimension ([11]) and corresponding to a valid ciphertext of the SIMD-product of the associated plaintexts $\mathbf{m}_1 \cdot \mathbf{m}_2$ with respect to a secret key sk' that is the tensor product of the secret key sk with itself. The Key Switching procedure ([27]) is then applied, using the matrix $W_{\mathsf{sk}^2 \to \mathsf{sk}}$, to obtain a valid ciphertext $\mathfrak{c}_{\mathsf{Mult}} \in R_{q_\mathfrak{l}}^2$ with respect to the original secret key sk. The noise magnitude in $\mathfrak{c}_{\mathsf{Mult}}$ is at approximately product of norms of the noise in $\mathfrak{c}_1'$ and $\mathfrak{c}_2'$.
     - If $\mathfrak{l} = 1$ then one applies the tensor operation to $\mathfrak{c}_1$ and $\mathfrak{c}_2$ directly, then the key switching is performed and only then is a SHE.LowerLevel operation performed. This results in us needing a larger prime p_1 than one would otherwise need, but more importantly a smaller p_0.
   - $\mathsf{SHE.ScalarMult}_{\mathsf{ek}}((\mathfrak{c}, \mathfrak{l}), \mathbf{a})$: If $\mathfrak{c} = (c_0, c_1)$ then one can obtain a homomorphic scalar multiplication by evaluating $\mathfrak{c}' = (\mathbf{a} \cdot c_0, \mathbf{a} \cdot c_1)$. This procedure increases the noise, but not by as much as a normal multiplication. Therefore we shall ignore the noise increase produced by scalar multiplication in our analysis.

Using the evaluation key we can also define an addition homomorphic operation as in [26, 27],

- $\mathsf{SHE.Permute}_{\mathsf{ek}}((\mathfrak{c}, \mathfrak{l}), \sigma) \rightarrow (\hat{\mathfrak{c}}_{\mathsf{Permute}}, \mathfrak{l})$: Given $\sigma \in \mathsf{Gal}$ and a ciphertext $\mathfrak{c} = (c_0, c_1) \in R_{q_\mathfrak{l}}^2$, corresponding to a plaintext $\mathbf{m} \in R_p$, this generates a ciphertext $\hat{\mathfrak{c}}_{\mathsf{Permute}} = (\hat{c}_0, \hat{c}_1) \in R_{q_\mathfrak{l}}^2$ corresponding to $\sigma(m)$, with respect to the secret key $\sigma(\mathsf{sk})$. Key switching is then applied, using the keyswitching matrix $W_{\sigma(\mathsf{sk}) \to \mathsf{sk}}$ to produce a ciphertext, $\hat{\mathfrak{c}}_{\mathsf{Permute}}$ decryptable under sk.

B.3 Defining SHE.Pack and SHE.Unpack for BGV

Despite our scheme being a packed SHE scheme it can still evaluate unpacked ciphertexts; indeed many of the instances of packed SHE schemes were originally conceived in the unpacked case by taking the map χ to be $\chi(m) = (m, m, \ldots, m)$, i.e. the *diagonal embedding*. For example this is the case with the schemes in [37, 11, 10, 8] etc all of which have packed counterparts. However, such a choice of χ is not efficient if one is interested in packing and unpacking encryptions of elements in $\mathbb{F}_p$. We wish to define two functions SHE.Pack and SHE.Unpack; the first of which takes N ciphertexts $\mathfrak{c}_i$ at level $\mathfrak{l}_i$ with the associated plaintext vector $\chi(m_i)$ for $m_i \in \mathbb{F}_p$, and produces a single ciphertext $\mathfrak{c}$ at level $\min(\mathfrak{l}_i)$ with the associated plaintext vector $\mathbf{m} = (m_1, \ldots, m_N) \in \mathcal{M}$. The second function performs the reverse operation.

In what follows we let $\mathbf{e}_i$ denote the i-th unit vector in $\mathcal{M}$, i.e. the element which is zero except for a one in the i-th position. To ease notation we let $\oplus$ and $\otimes$ denote the operations of applying the SHE.Add and SHE.Mult/SHE.ScalarMult operations respectively, we also let $\sigma(\mathfrak{c})$ denote applying the SHE.Permute operation to a ciphertext $\mathfrak{c}$ and map $\sigma \in \mathsf{Gal}$. If we define χ by the diagonal embedding then SHE.Pack can be defined in the following way

$$\mathsf{SHE.Pack}(\mathfrak{c}_1, \ldots, \mathfrak{c}_N) = \bigoplus_{i=1}^{N} \mathbf{e}_i \otimes \mathfrak{c}_i,$$

i.e. $\mathsf{SHE.Pack}$ is an $O(N)$ operation. However, $\mathsf{SHE.Unpack}$ needs to be performed as follows for $i = 1, \ldots, N$,

$$\mathsf{SHE.Unpack}(\mathfrak{c}) = \left(\bigoplus_{j=1}^{N} \sigma_{1 \to j}(\mathbf{e}_1 \otimes \mathfrak{c}), \ldots, \bigoplus_{j=1}^{N} \sigma_{N \to j}(\mathbf{e}_N \otimes \mathfrak{c}) \right)$$

i.e. $\mathsf{SHE.Unpack}$ is an $\mathcal{O}(N^2)$ operation. On the other hand, if we define χ to be the map $\chi(m) = (m, 0, \ldots, 0)$ then we can define $\mathsf{SHE.Pack}$ and $\mathsf{SHE.Unpack}$ by the following $\mathcal{O}(N)$ operations;

$$\mathsf{SHE.Pack}(\mathfrak{c}_1, \ldots, \mathfrak{c}_N) = \bigoplus_{i=1}^{N} \sigma_{i \to j}(\mathfrak{c}_i), \quad \mathsf{SHE.Unpack}(\mathfrak{c}) = (\mathbf{e}_1 \otimes \mathfrak{c}, \sigma_{2 \to 1}(\mathbf{e}_2 \otimes \mathfrak{c}), \ldots, \sigma_{N \to 1}(\mathbf{e}_N \otimes \mathfrak{c})) \, .$$

Thus we will utilize the mapping $\chi(m) = (m, 0, \ldots, 0)$ in our proposal.

B.4 Distributed Decryption Protocol

All that remains to define our Threshold L-levelled Packed SHE system based on BGV is to present the distributed decryption protocol. Note that we do not use the key-homomorphic properties of RLWE schemes as previously used in [1, 33, 2]. Instead, we follow the approach of [5], where the authors construct a threshold variant of Regev's cryptosystem ([36]); we adapt this method to our situation.

At a high level the method works as follows: we modify the $\mathsf{SHE.KeyGen}$ algorithm so that it also outputs for each party P_i a key dk_i for performing distributed decryption. The key dk_i consists of two components; i.e. $\mathsf{dk}_i = (\mathsf{sk}_i, \mathbf{k}_i)$. The values sk_i form a Shamir sharing over the ring R_{q_0} of the secret key sk, with threshold t. The value $\mathbf{k}_i$ are the associated keys for the PRSS described above. Given a common ciphertext $\mathfrak{c} = (c_0, c_1) \in R_{q_\mathfrak{l}}$ as input (for decryption), the parties first apply $\mathsf{SHE.LowerLevel}$ to reduce the ciphertext to level zero. Then each party P_i computes a decryption share $\bar{\mu}_i$ using his private sk_i and a PRSS over R_{q_0} as described earlier. The underlying PRF we assume produces values in the range

$$\left[- \frac{(2^{\mathsf{exp}} - 1) \cdot B_{\mathsf{dec}}}{p \cdot \binom{n}{t}}, \frac{(2^{\mathsf{exp}} - 1) \cdot B_{\mathsf{dec}}}{p \cdot \binom{n}{t}} \right],$$

where B_{dec} is the bound on the canonical norm of an element being decrypted mentioned earlier, and hence an upper bound on the size of the coefficients of the noise polynomial reconstructed during the standard decryption procedure. See Appendix C for a detailed discussion of B_{dec}. The choice of this range of the underlying PRF family means that the values output by the PRSS will be shares of elements in $R_{q_\mathfrak{l}}$ whose coefficients lie in the range $[-(2^{\mathsf{exp}} - 1) \cdot B_{\mathsf{dec}}/p, (2^{\mathsf{exp}} - 1) \cdot B_{\mathsf{dec}}/p]$. Note, that $\binom{n}{t}$ for $t \approx n/2$ grows very fast, and so the for the above range of the PRF to be suitably large we require that n is small. In our discussion we implicitly assume n to be small, say $n < 10$.

Recall distributed decryption is defined by two algorithms $\mathsf{SHE.ShareDec}$ and $\mathsf{SHE.ShareCombine}$. These are defined by the following procedures:

– $\mathsf{SHE.ShareDec}_{\mathsf{dk}_i}((\mathfrak{c}, \mathfrak{l})) \to \bar{\mu}_i$:

1. Apply $\mathsf{SHE.LowerLevel}_{\mathsf{ek}}((\mathfrak{c}, \mathfrak{l}), 0)$ to obtain the ciphertext (c_0, c_1) at level zero (unless $\mathfrak{c}$ is already at level zero).
2. Compute $\mu_i = [\mu]_i = [c_0 - \mathsf{sk} \cdot c_1]_i = c_0 - \mathsf{sk}_i \cdot c_1$ where the computation is in R_{q_0}.
3. Execute the PRSS, using the PRF keys $\mathbf{k}_i$, to obtain a Shamir's share r_i of a "random" value $r \in R_{q_0}$ such that $r = \sum_{\mathbf{k}_A} \psi_{\mathbf{k}_A}(\mathbf{t})$ and $\|r\|_\infty < ((2^{\mathsf{exp}} - 1) \cdot B_{\mathsf{dec}})/p$, for some agreed vector of values $\mathbf{t}$ which are a function of the input ciphertext $\mathfrak{c}$.
4. Compute $\bar{\mu}_i = [\bar{\mu}]_i = [\mu + p \cdot r]_i = \mu_i + p \cdot r_i$ and output $\bar{\mu}_i$ as the decryption share.

– $\mathsf{SHE.ShareCombine}((\mathfrak{c}, \mathfrak{l}), \{\bar{\mu}_i\}_{i \in [1,\ldots,n]}) \to \mathbf{m}'$: given a set n of decryption shares $\bar{\mu}_i$ and (in the malicious setting) an error correction procedure, reconstruct $\bar{\mu} = \mu + p \cdot r$ by applying the error correction procedure to $\{\bar{\mu}_i\}_{i \in [1\ldots,n]}$ and output $\mathbf{m}' = (\bar{\mu} \mod p)$.

Note decryption will work as long as the reconstructed value $\bar{\mu}$ is less than $q_0/2$, i.e. we require $2^{\mathsf{exp}} \cdot B_{\mathsf{dec}} < q_0/2$ (see the next section for details).

We pause to note the different situations where one obtains correct message recovery from the algorithm SHE.ShareCombine. In the case of passive adversaries we will show (in the next section) that the above distributed decryption procedure is secure as long as $t < n$. Since we are using Shamir sharing, in the presence of $t < n/3$ active corruptions, using the natural error correction properties (namely Reed-Solomon (RS) error correction), we can correctly recover the message at the end of SHE.ShareCombine.

When $t < n/2$ a little more work is involved; if an adversary sends an incorrect share then this can be detected, again because we are using Shamir as the underlying secret sharing scheme. At this point the parties execute a party elimination strategy in which they require each other to prove in zero-knowledge that the provided share is correct. Once the cheater party(s) have been determined they are eliminated from the protocol and the protocol resumes. Thus for active adversaries and $t < n/2$ we may require a grand total of an extra $n^2 \cdot t$ zero-knowledge proofs to be constructed, irrespective of the size of the circuit in our main protocol; see Appendix F for more details.

B.5 Security of Our Threshold BGV Instantiation

Recall from earlier we require four security properties:

– Key Generation Security.
– Semantic Security.
– Correct Share Decryption.
– Share Simulation Indistinguishability.

We now discuss each of these in turn.

Key Generation Security: The required properties of the keys produced by the key generation algorithm follow from the security properties of the Shamir secret sharing scheme used to share sk. We note in our main protocol we assume an ideal functionality to distribute such keys, and so there is no "Key Generation" protocol to analyse.

Semantic Security: The follows from the standard semantic security of the BGV scheme. However, we need to deal with the fact that the adversary has access to shares of the underlying secret key and the keys to the PRSS. A standard simulation shows that security in our setting reduces to that in the standard setting.

Correct Share Decryption: The infinity norm of the element $\bar{\mu} = \mu + p \cdot r$ produced by the algorithm SHE.ShareCombine is bounded by $B_{\mathsf{dec}} + p \cdot (2^{\mathsf{exp}} - 1) \cdot B_{\mathsf{dec}}/p \approx 2^{\mathsf{exp}} \cdot B_{\mathsf{dec}}$. If $2^{\mathsf{exp}} \cdot B_{\mathsf{dec}} < q_0/2$ then correct decryption will result.

Share Simulation Indistinguishability: We need to present a PPT algorithm (simulator) $\mathsf{SHE.ShareSim}$ which when given a ciphertext $\mathfrak{c} \in R_{q_\mathfrak{l}}^2$ with associated plaintext $\mathbf{m} \in R_p$, a subset $I \subset \{1, \ldots, n\}$ such that $|I| = t$, and a set of t decryption shares $\{\bar{\mu}_i\}_{i \in I}$, where $\bar{\mu}_i = \mathsf{SHE.ShareDec}_{\mathsf{dk}_i}((\mathfrak{c}, \mathfrak{l}))$, can simulate the remaining $(n - t)$ decryption shares $\{\bar{\mu}_j^*\}$ in such a way that the following two following distributions are statistically indistinguishable:

$$\left(\{\bar{\mu}_i\}_{i \in I}, \{\bar{\mu}_j^*\}_{j \in \bar{I}}\right) \overset{s}{\approx} \left(\{\bar{\mu}_i\}_{i \in I}, \{\bar{\mu}_j\}_{j \in \bar{I}}\right),$$

where $\bar{I} = \{1, \ldots, n\} \setminus I$. i.e. one cannot distinguish the real shares for the set $\bar{I}$ (as computed by $\mathsf{SHE.ShareDec}$ algorithm) with ones produced by the simulator. Moreover, we require the statistical distance between the two distributions to be bounded by $2^{-\mathsf{sec}}$. The simulator is constructed as follows:

1. Let $\mathbf{k}_A^{(I)}$ denote the set of keys for the PRSS that have been given to parties P_i where $i \in I$, and let $\mathbf{k}_A^{(\bar{I})}$ denote the set of keys for the PRSS held by P_j, for $j \in \bar{I}$.
2. The simulator first computes
$$r' = \sum_{\mathbf{k} \in \mathbf{k}_A^{(I)}} \psi_{\mathbf{k}}(\mathbf{t}) + \sum_{\mathbf{k} \in \mathbf{k}_A^{(\bar{I})}} r_{\mathbf{k}},$$
where each $r_{\mathbf{k}} \in R_{q_\mathfrak{l}}$ is chosen such that
$$\|r_{\mathbf{k}}\|_\infty < \frac{(2^{\mathsf{exp}} - 1) \cdot B_{\mathsf{dec}}}{p \cdot \binom{n}{t}}.$$
In this way $\|r'\|_\infty < \frac{(2^{\mathsf{exp}}-1) \cdot B_{\mathsf{dec}}}{p}$.
3. Let $\bar{\mu}^* = \mathbf{m} + p \cdot r'$. For each $j \in \bar{I}$, the simulator outputs $\bar{\mu}_j^*$ such that $\left(\{\bar{\mu}_j^*\}_{j \in \bar{I}}, \{\bar{\mu}_i\}_{i \in I}\right)$ is a consistent vector of shares of $\bar{\mu}^*$; *i.e.* the simulator deterministically computes consistent shares for the honest parties via Lagrange interpolation of the $t + 1$ values, $\bar{\mu}^*$ and $\{\bar{\mu}_i\}_{i \in I}$.

Before proving the properties of the simulation, we recall the following lemma from [2]:

Lemma 1 (Smudging Lemma [2]). *Let B_1 and B_2 be positive integers and let $e_1 \in [-B_1, B_1]$ be a fixed integer and let $e_2 \in [-B_2, B_2]$ be chosen uniformly and randomly. Then the statistical distance between the distribution of e_2 and $e_2 + e_1$ is B_1/B_2.*

To prove the properties of the simulation, we first note that similar to the last stage of the simulation above, the real shares for the honest parties can be constructed (deterministically) from $\bar{\mu}$ and the shares held by the t dishonest parties. Thus, to prove indistinguishability of the real and simulated shares, it suffices to prove that $\bar{\mu}^* = \mathbf{m} + p \cdot r'$ and $\bar{\mu} = \mu + p \cdot r$ are statistically close[3]. To see this is indeed the case, we first note that $\mu + p \cdot r$ and $\mu + p \cdot r'$ are indistinguishable (by construction) and that r' is uniform in an exponentially larger range than μ (recall that $\|\mu\|_\infty < B_{\mathsf{dec}}$ and $\|r'\|_\infty < \frac{(2^{\mathsf{exp}}-1) \cdot B_{\mathsf{dec}}}{p}$). By application of the Smudging lemma, the statistical distance between the distribution of $\mu + p \cdot r'$ and the uniform distribution of polynomials with coefficients in $[-(2^{\mathsf{exp}} - 1) \cdot B_{\mathsf{dec}}, (2^{\mathsf{exp}} - 1) \cdot B_{\mathsf{dec}}]$ is exactly $N/(2^{\mathsf{exp}} - 1)$.

To conclude the proof, we next claim that the distribution of $\mathbf{m} + p \cdot r'$ is statistically indistinguishable from the uniform distribution of polynomials with coefficients $[-(2^{\mathsf{exp}} - 1) \cdot B_{\mathsf{dec}}, (2^{\mathsf{exp}} - 1) \cdot B_{\mathsf{dec}}]$. This follows from the fact that the statistical distance between the two distributions is $\frac{N \cdot p}{B_{\mathsf{dec}} \cdot (2^{\mathsf{exp}}-1)}$ (which itself

[3] For statistically close distributions $X \overset{s}{\approx} Y$ and any deterministic procedure A applied to those distributions it is the case that $A(X) \overset{s}{\approx} A(Y)$.

follows from the Smudging Lemma and the fact that $\mathbf{m} \in R_p$). It follows from the triangle inequality that the overall statistical distance between the distribution of $\bar{\mu}^* = \mathbf{m} + p \cdot r'$ and $\bar{\mu} = \mu + p \cdot r$ is upper bounded by $\frac{N \cdot (p + B_{\mathsf{dec}})}{B_{\mathsf{dec}} \cdot (2^{\mathsf{exp}} - 1)}$. Choose

$$\mathsf{exp} = \mathsf{sec} + \max\left(\log_2\left(\frac{N \cdot (p + B_{\mathsf{dec}})}{B_{\mathsf{dec}}} \right), \log_2(N) \right).$$

Since $p < B_{\mathsf{dec}}$ this simplifies to $\mathsf{exp} = \mathsf{sec} + \log_2(N) + 1$, and we can therefore ensure the statistical distance is bounded by $2^{-\mathsf{sec}}$ which can be made arbitrarily small by our choice of exp.

B.6 Batch Distributed Decryption

Using a well known technique presented in [3, 18], we can perform a batch of $t + 1 = \Theta(n)$ distributed decryption, and hence evaluate a batch of $t + 1$ Refresh gates at the communication cost of performing two distributed decryptions. The following technique applies to our main MPC protocol if the batch of refresh gates are *independent*, meaning the output wire of one does not lead to the input of the other.

Given a value shared among the parties, its public reconstruction requires each party to send the share (of the value) it holds to every other party. This requires $n \cdot (n - 1)$ pair-wise communication of shares. So for $t + 1$ shared values, the public reconstruction will require $\mathcal{O}(n^3)$ pair-wise communication of shares. In what follows, it is shown how the above can be achieved with the same cost of public reconstruction of a single value, namely with a communication of $2 \cdot n \cdot (n - 1) = \mathcal{O}(n^2)$ shares. The idea was used in the information theoretically secure MPC protocols of [3, 18].

Let $u^{(1)}, \ldots, u^{(t+1)}$ be $t + 1$ shared values. First the $t + 1$ shared values are "expanded" to n shared values, say $v^{(1)}, \ldots, v^{(n)}$ by applying a linear function locally. Specifically, if the underlying LSS is Shamir, then we can interpret $u^{(1)}, \ldots, u^{(t+1)}$ as the coefficients of a polynomial of degree at most t, say $u(\cdot)$ and let $v^{(1)}, \ldots, v^{(n)}$ be the n distinct points on this polynomial. Now notice that obtaining $v^{(1)}, \ldots, v^{(n)}$ from $u^{(1)}, \ldots, u^{(t+1)}$ is a linear function and by (locally) applying the same linear function on the sharings of $u^{(1)}, \ldots, u^{(t+1)}$, the parties can obtain sharings of $v^{(1)}, \ldots, v^{(n)}$. Now each $v^{(i)}$ is reconstructed only to P_i and this costs $\mathcal{O}(n^2)$ communication of shares. Finally every P_i sends $v^{(i)}$ to every other party (which costs another $\mathcal{O}(n^2)$ communication) and then every party can reconstruct $u(\cdot)$ and hence $u^{(1)}, \ldots, u^{(t+1)}$.

In our setting all of the above sharing is done using Shamir over the ring R_{q_0}. It is easy to see that the above can be carried out with no change to the underlying SHE scheme. Thus assuming our initial circuit is large enough, i.e. there are enough independent Refresh gates at each level, we can obtain a performance improvement of $(t + 1)/2$.

C Parameter Calculation

In [27] a concrete set of parameters for the BGV SHE scheme was given for the case of binary message spaces, and arbitrary L. In [16] this was adapted to the case of message space R_p for 2-power cyclotomic rings, but only for the schemes which could support one level of multiplication gates (i.e. for $L = 1$). In this section we combine these analyses to produce parameter estimations for the case we require of arbitrary L and messages defined by a "large prime", e.g. $p \approx 2^{32}$, 2^{64} or 2^{128}. We assume in this section that the reader is familiar with the analysis and algorithms from [27]; we mainly point out the differences in estimates for our case.

Our analysis will make extensive use of the following fact: If $a \in R$ be chosen from a distribution such that the coefficients are distributed with mean zero and standard deviation σ, then if ζ_m is a primitive mth root of unity, we can use $6 \cdot \sigma$ to bound $a(\zeta_m)$ and hence the canonical embedding norm of a. If we have two elements with variances σ_1^2 and σ_2^2, then we can bound the canonical norm of their product with $16 \cdot \sigma_1 \cdot \sigma_2$.

Recall from Appendix B that we require a chain of moduli $q_0 < q_1 \ldots < q_L$ corresponding to each level of the scheme, where $q_L = q_0 \cdot \prod_{i=1}^{i=L} p_i$. Note that we evaluate a depth L circuit from a chain of $L + 1$ moduli. Also note, that we apply a SHE.LowerLevel (a.k.a. modulus switch) algorithm *before* a multiplication operation, except when multiplying at level one. This often leads to lower noise values in practice (which a practical instantiation can make use of). In addition it eliminates the need to perform a modulus switch after encryption.

We utilize the following constants described in [16], which are worked out for the case of message space defined modulo p (the constants in [16] make use of an additional parameter n, arising from the key generation procedure. In our case we can take this constant equal to one).

$$\begin{aligned} B_{\mathsf{Clean}} =& N \cdot p/2 + p \cdot \sigma \cdot \left(\frac{16 \cdot N}{\sqrt{2}} + 6 \cdot \sqrt{N} + 16 \cdot \sqrt{h \cdot N} \right) \\ B_{\mathsf{Scale}} =& p \cdot \sqrt{3 \cdot N} \cdot \left(1 + \frac{8}{3} \cdot \sqrt{h} \right) \\ B_{\mathsf{Ks}} =& p \cdot \sigma \cdot N \cdot \left(1.49 \cdot \sqrt{h \cdot N} + 2.11 \cdot h + 5.54 \cdot \sqrt{h} + 1.96\sqrt{N} + 4.62 \right) \end{aligned}$$

The constants are used in the following manner: A freshly encrypted ciphertext at level L has noise bounded by B_{Clean}. In the worst case, when applying SHE.LowerLevel to a ciphertext at level $\mathfrak{l}$ with noise bounded by B' one obtains a new ciphertext at level $\mathfrak{l} - 1$ with noise bounded by

$$\frac{B'}{p_{\mathfrak{l}}} + B_{\mathsf{Scale}}.$$

When applying the tensor product multiplication operation to ciphertexts of a given level $\mathfrak{l}$ of noise B_1 and B_2 one obtains a new ciphertext with noise given by

$$B_1 \cdot B_2 + \frac{B_{\mathsf{Ks}} \cdot q_{\mathfrak{l}}}{P} + B_{\mathsf{Scale}},$$

where P is a value to be determined later. As in [27] we define a small "wiggle room" ξ which we set to be equal to eight; this is set to enable a number of additions to be performed without needing to individually account for them in our analysis.

A general evaluation procedure begins with a freshly encrypted ciphertext at level L with noise B_{Clean}. When entering the first multiplication operation we first apply a SHE.LowerLevel operation to reduce the noise to our universal bound B. We therefore require

$$\frac{\xi \cdot B_{\mathsf{Clean}}}{p_L} + B_{\mathsf{Scale}} \leq B,$$

i.e.

$$p_L \geq \frac{8 \cdot B_{\mathsf{Clean}}}{B - B_{\mathsf{Scale}}}. \tag{1}$$

We now turn to dealing with the SHE.LowerLevel operation which occurs before a multiplication gate at level $\mathfrak{l} \in [2, \ldots, L - 1]$. We perform a worst case analysis and assume that the input ciphertexts are at level $\mathfrak{l}-1$. We can then assume that the input to the tensoring operation in the previous multiplication gate (just after the previous $\mathsf{SHE.LowerLevel}$) was bounded by B, and so the output noise from the previous multiplication gate for each input ciphertext is bounded by $B^2 + B_{\mathsf{Ks}} \cdot q_{\mathfrak{l}}/P + B_{\mathsf{Scale}}$. This means the noise on entering the $\mathsf{SHE.LowerLevel}$ operation is bounded by ξ times this value, and so to maintain our invariant we require

$$\frac{\xi \cdot B^2 + \xi \cdot B_{\mathsf{Scale}}}{p_{\mathfrak{l}}} + \frac{\xi \cdot B_{\mathsf{Ks}} \cdot q_{\mathfrak{l}}}{P \cdot p_{\mathfrak{l}}} + B_{\mathsf{Scale}} \leq B.$$

Rearranging this into a quadratic equation in B we have

$$\frac{\xi}{p_{\mathfrak{l}}} \cdot B^2 - B + \left(\frac{\xi \cdot B_{\mathsf{Scale}}}{p_{\mathfrak{l}}} + \frac{\xi \cdot B_{\mathsf{Ks}} \cdot q_{\mathfrak{l}-1}}{P} + B_{\mathsf{Scale}} \right) \leq 0.$$

We denote the constant term in this equation by $R_{\mathfrak{l}-1}$. We now assume that all primes $p_{\mathfrak{l}}$ are of roughly the same size, and noting the we need to only satisfy the inequality for the largest modulus $\mathfrak{l} = L-1$. We now fix R_{L-2} by trying to ensure that R_{L-2} is close to $B_{\mathsf{Scale}} \cdot (1 + \xi/p_{L-1}) \approx B_{\mathsf{Scale}}$, so we set $R_{L-2} = (1-2^{-3}) \cdot B_{\mathsf{Scale}}(1+\xi/p_{L-1})$, and obtain

$$P \approx 8 \cdot \frac{\xi \cdot B_{\mathsf{Ks}} \cdot q_{L-2}}{B_{\mathsf{Scale}}}, \tag{2}$$

since $B_{\mathsf{Scale}} \cdot (1+\xi/p_{L-1}) \approx B_{\mathsf{Scale}}$.

To ensure we have a solution we require $1 - 4 \cdot \xi \cdot R_{L-2}/p_{L-1} \geq 0$, which implies we should take, for $i = 2, \ldots, L-1$,

$$p_i \approx 4 \cdot \xi \cdot R_{L-2} \approx 32 \cdot B_{\mathsf{Scale}}. \tag{3}$$

Recall that the final multiplication is executed in a different manner. We do not modulus switch before the multiplication, but afterwards. We analyse the implication of this, for the size of p_1, from the point of view of our concrete application to our MPC protocol. The final multiplication will be of a ciphertext with noise

$$\xi \cdot (B^2 + B_{\mathsf{Scale}}) + \frac{\xi \cdot B_{\mathsf{Ks}} \cdot q_1}{P},$$

and a ciphertext with noise B (namely $\mathfrak{c}_1$). The input to the final key switch will have noise value approximately $\xi \cdot B^3$; we make this simplifying assumption which makes little difference to the final values. The output noise from the keyswitch is then equal to

$$\xi \cdot B^3 + B_{\mathsf{Scale}} + \frac{B_{\mathsf{Ks}} \cdot q_1}{P}.$$

We then perform a modulus switch to obtain a ciphertext as output of the multiplication gate with noise bounded by

$$\frac{\xi \cdot B^3 + B_{\mathsf{Scale}}}{p_1} + \frac{B_{\mathsf{Ks}} \cdot p_0}{P} + B_{\mathsf{Scale}}.$$

We again require this to be less than B, so we have now the cubic equation

$$\frac{\xi}{p_1} \cdot B^3 - B + \left(\frac{B_{\mathsf{Scale}}}{p_1} + \frac{B_{\mathsf{Ks}} \cdot p_0}{P} + B_{\mathsf{Scale}} \right) \leq 0.$$

Substituting in our existing estimate for P, namely $8 \cdot \xi \cdot B_{\mathsf{Ks}} \cdot q_{L-2}/B_{\mathsf{Scale}}$ we find the inequality is roughly equivalent to, assuming $L > 2$ and $p_1 \gg B_{\mathsf{Scale}}$ (i.e. $q_{L-2} \gg B_{\mathsf{Scale}} \cdot p_0$),

$$\frac{\xi}{p_1} \cdot B^3 - B + \frac{B_{\mathsf{Scale}}}{p_1} + B_{\mathsf{Scale}} \approx \frac{\xi}{p_1} \cdot B^3 - B + \left(\frac{B_{\mathsf{Scale}}}{p_1} + \frac{B_{\mathsf{Scale}} \cdot p_0}{8 \cdot q_{L-2}} + B_{\mathsf{Scale}} \right) \leq 0.$$

If we set $B \approx 2 \cdot B_{\mathsf{Scale}}$, then this means we have (approximately)

$$\frac{\xi}{p_1} \cdot 8 \cdot B_{\mathsf{Scale}}^3 - B_{\mathsf{Scale}} + \frac{B_{\mathsf{Scale}}}{p_1} \leq 0,$$

and so

$$p_1 \approx 8 \cdot (\xi + 1) \cdot B_{\mathsf{Scale}}^2 \tag{4}$$

will therefore guarantee the result.

We now need to estimate the size of p_0. Due to the above choice of p_1 the ciphertext to which we apply the distributed decryption has norm bound by B, to which we add on a random encryption of zero at level L. To do this we need to apply LowerLevel to this encryption of zero, and hence the noise level of the ciphertext we finally pass into $\mathsf{SHE.ShareDec}$ in our main MPC protocol has noise bounded by $B_{\mathsf{dec}} = 2 \cdot B$ This means that we require

$$q_0 = p_0 \geq 2^{\sec + 2} \cdot B, \tag{5}$$

to ensure a valid distributed decryption.

Finally, set the Hamming weight h of the secret key sk to be 64 as in [27, 16]. Plugging this into our equations (1), (2), (3), (4), and (5), we obtain

$$\begin{aligned} p_0 &\approx 309 \cdot 2^{\sec} \cdot p \cdot \sqrt{N}, \\ p_1 &\approx 107736 \cdot p^2 \cdot N, \\ p_i &\approx 1237 \cdot p \cdot \sqrt{N}, \text{ for } 2 \leq i \leq L-1, \\ p_L &\approx 2.34 \cdot \sigma \cdot \sqrt{N}, \\ P &\approx 0.404 \cdot 1237^L \cdot \sigma \cdot 2^{\mathsf{exp}} \cdot p^L \cdot N^{(L+3)/2}, \\ q_{L-1} &\approx 21.76 \cdot 1237^L \cdot 2^{\mathsf{exp}} \cdot p^{L+1} \cdot N^{(L+1)/2}. \end{aligned}$$

The largest modulus used in our key switching matrices, i.e. the largest modulus used in an LWE instance, is given by $Q_{L-1} = P \cdot q_{L-1}$; where using the above estimates we have

$$Q_{L-1} \approx 8.79 \cdot 1237^{2 \cdot L} \cdot \sigma \cdot 4^{\mathsf{exp}} \cdot p^{2 \cdot L + 1} \cdot N^{L+2}.$$

Recall from Appendix B.5 we have the following relationship between exp and our statistical security parameter sec; $\mathsf{exp} = \mathsf{sec} + \log_2(N)$. To ensure security we use the estimates of Lindner and Peikert [32], we require at the κ-bit security level we require

$$N > (\kappa + 110) \cdot \log(Q_{L-1}/\sigma)/7.2.$$

D The UC Security Model

We work in the standard Universal Composability (UC) framework of Canetti [12], with static corruption. The UC framework introduces a PPT environment $\mathcal{Z}$ that is invoked on the security parameter 1^κ and an auxiliary input z and oversees the execution of a protocol in one of the two worlds. The "ideal" world execution involves dummy parties $P_1, \ldots, P_n$, an ideal adversary $\mathcal{S}$ who may corrupt some of the dummy parties, and a functionality $\mathcal{F}$. The "real" world execution involves the PPT parties $P_1, \ldots, P_n$ and a real world adversary $\mathcal{A}$ who may corrupt some of the parties. In either of these two worlds, a PPT adversary can corrupt t parties out of the n parties. The environment $\mathcal{Z}$ chooses the input of the parties and may interact with the ideal/real adversary during the execution. At the end of the execution, it has to decide upon and output whether a real or an ideal world execution has taken place.

We let $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa, z)$ denote the random variable describing the output of the environment $\mathcal{Z}$ after interacting with the ideal execution with adversary $\mathcal{S}$, the functionality $\mathcal{F}$, on the security parameter 1^κ and z. Let $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa, z)\}_{\kappa\in\mathbb{N}, z\in\{0,1\}^*}$. Similarly let $\mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa, z)$ denote the random variable describing the output of the environment $\mathcal{Z}$ after interacting in a real execution of a protocol Π with adversary $\mathcal{A}$, the parties $\mathcal{P}$, on the security parameter 1^κ and z. Let $\mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa, z)\}_{\kappa\in\mathbb{N}, z\in\{0,1\}^*}$.

Definition 3. *For $n \in \mathbb{N}$, let $\mathcal{F}$ be an n-ary functionality and let Π be an n-party protocol. We say that Π securely realizes $\mathcal{F}$ if for every PPT real world adversary $\mathcal{A}$, there exists a PPT ideal world adversary $\mathcal{S}$, corrupting the same parties, such that the following two distributions are computationally indistinguishable:*

$$\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}.$$

We consider the above definition where it quantifies over different adversaries: passive or active, that corrupts only certain number of parties.

E Semi-honest Security

Proof of Theorem 1. We prove the theorem with respect to a generic L-levelled SHE scheme and first consider the correctness. Suppose in the protocol party P_i has input $x_i \in \mathbb{F}_p$. Then we claim the following invariant to hold for each wire w of the circuit C^{aug} during the execution of the protocol: if $(\mathfrak{c}, \mathfrak{l})$ is the ciphertext associated with w during the execution of the protocol where level $\mathfrak{l} \in [1, \ldots, L]$, then $\mathsf{SHE.Dec}_{\mathsf{sk}}(\mathfrak{c}, \mathfrak{l}) = \chi(z)$, where $z \in \mathbb{F}_p$ is the value that would have been associated with w during the evaluation of C^{aug} with input $\boldsymbol{x} = (x_1, \ldots, x_n)$. Before proving the claim, we first recall that due to the introduction of the $\mathsf{Refresh}$ gates in C^{aug} and the way circuit is evaluated, every wire in the circuit C^{aug} has label in the range $[1, \ldots, L]$ and the corresponding ciphertext associated with the wire (during the protocol execution) has level in the range $[1, \ldots, L]$. In addition the level of the ciphertext associated to a wire is equal to the label of the wire.

Our invariant is clearly true for the input wires. Assuming that the evaluation of the refresh gates is correct, the invariant is also true for the output of the $\mathsf{Refresh}$ gates. That the invariant holds for the rest of the circuit follows from the homomorphic property of the SHE scheme. Finally, the correctness of the refresh gate evaluation follows from the correctness of $\mathsf{SHE.Pack}$, $\mathsf{SHE.Unpack}$, the homomorphic of the underlying SHE; and the fact that all the ciphertexts that are used in evaluating a refresh gate have levels in the range $[0, \ldots, L]$.

We next prove the security. Let $\mathcal{A}$ be a real-world semi-honest adversary corrupting $t < n$ parties and let $T \subset \mathcal{P}$ denote the set of corrupted parties. We now present an ideal-world adversary (simulator) $\mathcal{S}_f^{\mathsf{SH}}$ for

$\mathcal{A}$ in Figure 6. The high level idea for the simulator is the following: the simulator takes the input $\{x_i\}_{P_i \in T}$ and interacts with $\mathcal{F}_f$ to obtain the function output y. The simulator then invokes $\mathcal{A}$ with the inputs $\{x_i\}_{P_i \in T}$ and simulates each message that $\mathcal{A}$ would have received in the protocol Π_f^{SH} from the honest parties and from the functionality $\mathcal{F}_{\text{SETUPGEN}}$, stage by stage.

Simulator $\mathcal{S}_f^{\text{SH}}$

Let SHE be an L-levelled SHE scheme. The simulator plays the role of the honest parties and simulates each step of the protocol Π_f^{SH} as follows. The communication of the $\mathcal{Z}$ with the adversary $\mathcal{A}$ is handled as follows: Every input value received by the simulator from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape. Likewise, every output value written by $\mathcal{A}$ on its output tape is copied to the simulator's output tape (to be read by the environment $\mathcal{Z}$). The simulator then does the following for the session ID sid:

Offline Computation:

- On receiving the message (sid, i) to $\mathcal{F}_{\text{SETUPGEN}}$ from $\mathcal{A}$ for each $P_i \in T$, the simulator invokes $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}, \mathsf{dk}_1, \ldots, \mathsf{dk}_n) = \mathsf{SHE.KeyGen}(1^\kappa, 1^{\mathsf{sec}}, n, t)$, computes $(\mathfrak{c}_\mathbf{0}, 1) = \mathsf{SHE.LowerLevel}_{\mathsf{ek}}(\mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{0}, \cdot), 1)$, and on the behalf of $\mathcal{F}_{\text{SETUPGEN}}$ sends $(\mathsf{sid}, \mathsf{pk}, \mathsf{ek}, \{\mathsf{dk}_i\}_{P_i \in T}, (\mathfrak{c}_\mathbf{0}, 1))$ to $\mathcal{A}$.
- For each $P_j \notin T$, the simulator computes $(\mathfrak{c}_{\mathbf{m}_{jk}}, L) = \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{m}_{j,k}, \cdot)$ for $k \in [1, \ldots, \zeta]$ for a randomly chosen $\mathbf{m}_{j,k} \in \mathcal{M}$ and sends $(\mathsf{sid}, j, (\mathfrak{c}_{\mathbf{m}_{j,1}}, L), \ldots, (\mathfrak{c}_{\mathbf{m}_{j,\zeta}}, L))$ to $\mathcal{A}$ on the behalf of the honest parties.
- On receiving $(\mathsf{sid}, i, (\mathfrak{c}_{\mathbf{m}_{i,1}}, L), \ldots, (\mathfrak{c}_{\mathbf{m}_{i,\zeta}}, L))$ from $\mathcal{A}$ for every $P_i \in T$, the simulator decrypts the ciphertexts to get their associated plaintexts $\mathbf{m}_{i,1}, \ldots, \mathbf{m}_{i,\zeta}$; i.e. $\mathbf{m}_{i,k} = \mathsf{SHE.Dec}_{\mathsf{sk}}(\mathfrak{c}_{\mathbf{m}_{i,k}}, L)$. The simulator then applies $\mathsf{SHE.Add}$ on $(\mathfrak{c}_{\mathbf{m}_{1,k}}, L), \ldots, (\mathfrak{c}_{\mathbf{m}_{n,k}}, L)$ and sets the resultant ciphertext as the kth offline ciphertext. Furthermore it sets $\mathbf{m}_k = \mathbf{m}_{1,k} + \ldots + \mathbf{m}_{n,k}$ as the kth offline plaintext.

Online Computation:

- **Input Stage**: For every party $P_j \in \mathcal{P} \setminus T$, the simulator computes a random encryption $(\mathfrak{c}_{\mathbf{x}_j}, 1) = \mathsf{SHE.LowerLevel}_{\mathsf{ek}}(\mathsf{SHE.Enc}_{\mathsf{pk}}(\chi(0), \cdot), 1)$ and sends $(\mathsf{sid}, j, (\mathfrak{c}_{\mathbf{x}_j}, 1))$ to $\mathcal{A}$ on the behalf of every $P_j \in \mathcal{P} \setminus T$. The simulator receives $(\mathsf{sid}, i, (\mathfrak{c}_{\mathbf{x}_i}, 1))$ from $\mathcal{A}$ and obtains the associated plaintext $\mathbf{x}_i$. On the behalf of the parties $P_i \in T$, the simulator sends (sid, i, x_i) to the functionality $\mathcal{F}_f$ and receives y, where $x_i = \chi^{-1}(\mathbf{x}_i) \in \mathbb{F}_p$,
- **Computation Stage**: The simulator performs the local computation (required for the addition, multiplication and refresh gates) as specified in the protocol in order to be synchronized with the adversary with respect to the ciphertexts associated with the wires in the circuit. For the refresh gates, the simulator simulates to $\mathcal{A}$ the communication from the honest parties as follows:
  - **Refresh Gate**: Let this be the kth refresh gate and let $(\mathfrak{c}_{\mathbf{m}_k}, L)$ be the kth offline ciphertext with the associated plaintext $\mathbf{m}_k$, which are known to the simulator while simulating the offline computation. Let $(\mathfrak{c}, 0)$ be the ciphertext obtained after the masking operation. Since $\mathfrak{c}_\mathbf{1}$ is replaced by $\mathfrak{c}_\mathbf{0}$ in the simulation, $\mathfrak{c}$ is associated with message $\mathbf{m}_k$. For each $P_i \in T$, on receiving $(\mathsf{sid}, i, \bar{\mu}_i)$ from $\mathcal{A}$ as the decryption shares of $(\mathfrak{c}, 0)$, the simulator computes the simulated decryption shares $\{\bar{\mu}_j^*\}_{P_j \notin T} = \mathsf{SHE.ShareSim}((\mathfrak{c}, 0), \mathbf{m}_k, \{\bar{\mu}_i\}_{P_i \in T})$. The simulator then sends the simulated shares $\{\bar{\mu}_j^*\}_{P_j \notin T}$ to $\mathcal{A}$ as the decryption shares on the behalf of the honest parties.
- **Output Stage**:
  - **Randomization**: On receiving $(\mathsf{sid}, i, (\mathfrak{c}_i, L))$ for every $P_i \in T$ from $\mathcal{A}$, the simulator computes encryptions of $\chi(0)$ for every honest party, except for one honest party, say P_h, it encrypts $\chi(y)$. The simulator sends these ciphertexts to $\mathcal{A}$ on the behalf of the honest parties and then follows the protocol steps to obtain $(\hat{\mathfrak{c}}, 0)$ corresponding to the output wire. Note that the plaintext associated with $\hat{\mathfrak{c}}$ is $\chi(y)$, since $\mathfrak{c}_\mathbf{1}$ is replaced by $\mathfrak{c}_\mathbf{0}$ in the simulation and one of the ciphertexts on the behalf of an honest party (for randomization) encrypts $\chi(y)$.
  - On receiving the decryption share $(\mathsf{sid}, i, \bar{\gamma}_i)$ for every $P_i \in T$ from $\mathcal{A}$, the simulator computes the simulated decryption shares $\{\bar{\gamma}_j^*\}_{P_j \in \mathcal{P} \setminus T} = \mathsf{SHE.ShareSim}((\hat{\mathfrak{c}}, 0), \chi(y), \{\bar{\gamma}_i\}_{P_i \in T})$ for the the honest parties $P_j \in \mathcal{P} \setminus T$ and sends $(\mathsf{sid}, j, \bar{\gamma}_j^*)$ as the decryption shares to $\mathcal{A}$.

The simulator then outputs $\mathcal{A}$'s output.

Fig. 6. Simulator for the semi-honest adversary $\mathcal{A}$ corrupting t parties in the set $T \subset \mathcal{P}$.

We will now prove that $\text{IDEAL}_{\mathcal{F}_f,\mathcal{S}_f^{\text{SH}},\mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\Pi_f^{\text{SH}},\mathcal{A},\mathcal{Z}}$ via a series of hybrids. The output of each hybrid is always just the output of the environment $\mathcal{Z}$. Starting with $\mathbf{HYB}_0 = \text{REAL}_{\Pi_f^{\text{SH}},\mathcal{A},\mathcal{Z}}$, we gradually make changes to define $\mathbf{HYB}_1$, $\mathbf{HYB}_2$, $\mathbf{HYB}_3$ and $\mathbf{HYB}_4$.

$\mathbf{HYB}_1$**:** Same as $\mathbf{HYB}_0$, except that the decryption shares of the honest parties corresponding to the ciphertext $\hat{\mathfrak{c}}$ associated with the output wire (obtained after the randomization) are computed using SHE.ShareSim, by inputting to it the decryption shares of the corrupted parties corresponding to $\hat{\mathfrak{c}}$, the ciphertext $\hat{\mathfrak{c}}$ and the plaintext $\chi(y)$, where y is the function output.

$\mathbf{HYB}_2$**:** Same as $\mathbf{HYB}_1$, except that $\mathfrak{c}_\mathbf{1}$ obtained from $\mathcal{F}_{\text{SETUPGEN}}$ is replaced by $\mathfrak{c}_\mathbf{0}$ and the circuit is computed as in protocol with $\mathfrak{c}_\mathbf{0}$ being used in place of $\mathfrak{c}_\mathbf{1}$. Moreover, during the randomization step while performing the distributed decryption of the output wire ciphertext, the randomizing ciphertext $(\mathfrak{c}_i, L)$ of *one* of the *honest* parties (which is an encryption of $\mathbf{0}$), say P_h, is replaced by a random encryption of $\chi(y)$.

$\mathbf{HYB}_3$**:** Same as $\mathbf{HYB}_2$, except that SHE.ShareSim is used while computing the decryption shares of the honest parties for performing the distributed decryption during the evaluation of the refresh gates.

$\mathbf{HYB}_4$**:** Same as $\mathbf{HYB}_3$, except that the real inputs of the honest parties are replaced by $\chi(0)$ during the **Input Stage** and the circuit is evaluated using encryptions of the $\chi(0)$s as the encrypted inputs of the honest parties.

Our proof will conclude, as we show that every two consecutive hybrids are computationally indistinguishable and $\mathbf{HYB}_4 = \text{IDEAL}_{\mathcal{F}_f,\mathcal{S}_f^{\text{SH}},\mathcal{Z}}$.

$\mathbf{HYB}_0 \overset{c}{\approx} \mathbf{HYB}_1$**:** This follows from the share simulation indistinguishability property of SHE.

$\mathbf{HYB}_1 \overset{c}{\approx} \mathbf{HYB}_2$**:** To show the indistinguishability, we rely on the semantic security of SHE. In fact, we use a variant of the semantic security notion, where the adversary gives two pairs of messages to the challenger and the challenger picks a random pair and gives the encryptions for that pair to the adversary. We call this as the *double message* semantic security. It follows by a standard hybrid argument that a scheme offering semantic security also offers double message semantic security with a security loss of a factor of two.

We now show how a distinguisher $\mathcal{Z}$ for the hybrids $\mathbf{HYB}_1$ and $\mathbf{HYB}_2$ can be used to break the double message semantic security of the underlying SHE. Let $\mathcal{R}$ be the attacker that wants to break the double message semantic security of the underlying SHE; $\mathcal{R}$ uses $\mathcal{Z}$ to do so as follows: $\mathcal{R}$ receives the public key pk, evaluation key ek and t keys corresponding to the corrupted parties for performing the distributed decryption. The attacker $\mathcal{R}$ then invokes $\mathcal{Z}$ (in her head), which gives back the input set $(x_1, \ldots, x_n) \in \mathbb{F}_p^n$ for all the parties. Using this output $\mathcal{R}$ computes the function output y and prepares two pairs of messages for the challenger, $(\mathbf{1}, \mathbf{0})$ and $(\mathbf{0}, \chi(y))$ and hands them over to the challenger. Let $\mathcal{R}$ receive back the encrypted pair $(\mathfrak{c}', L), (\mathfrak{c}, L)$ from the challenger. The algorithm $\mathcal{R}$ now applies SHE.LowerLevel to reduce the first of these to level one, (by abuse of notation we shall still refer to it as $\mathfrak{c}'$). Now $\mathcal{R}$ evaluates the circuit by generating offline data honestly and using $(\mathfrak{c}', 1)$ in place of $(\mathfrak{c}_\mathbf{1}, 1)$ (that was to be returned by $\mathcal{F}_{\text{SETUPGEN}}$) and $(\mathfrak{c}, L)$ in place of the randomization ciphertext (namely an encryption of $\mathbf{0}$) on the behalf of the honest party P_h (which P_h would have given to randomize the output wire ciphertext). Finally $\mathcal{R}$ outputs what $\mathcal{Z}$ outputs.

It is easy to note that if the challenger had given encryptions of the first pair of messages, namely $(\mathbf{1}, \mathbf{0})$, then $\mathcal{Z}$ is in $\mathbf{HYB}_1$, else it is in $\mathbf{HYB}_2$. Thus the distinguishing probability of $\mathcal{Z}$ is translated to the winning probability of $\mathcal{R}$ in the double message semantic security game. This implies that our claim is true and there exists no PPT distinguisher $\mathcal{Z}$ for the above two hybrids.

$\mathbf{HYB}_2 \overset{c}{\approx} \mathbf{HYB}_3$**:** This can be shown by relying on the share simulation indistinguishability property of SHE and by defining $\mathbb{G}_R$ hybrids over the number of refresh gates, where the ith hybrid is same as $\mathbf{HYB}_2$, except that SHE.ShareSim is invoked for the first i refresh gates (assuming topological ordering of the gates) to compute the decryption shares of the honest parties and for the $(i+1)$th refresh gate onwards, the decryption shares of the honest parties are computed as in real protocol using SHE.ShareDec.

$\mathbf{HYB}_3 \overset{c}{\approx} \mathbf{HYB}_4$**:** We resort to the semantic security of the underlying SHE scheme. We let $H = |\mathcal{P} \setminus T|$ denote the number of honest parties and without loss of generality assume that the first H parties are the honest parties. We introduce $H+1$ hybrids $\mathbf{HYB}_3^0 = \mathbf{HYB}_3, \mathbf{HYB}_3^1, \ldots, \mathbf{HYB}_3^H = \mathbf{HYB}_4$ over the number of honest parties so that the ith hybrid $\mathbf{HYB}_3^i$ is same as the $(i-1)$th hybrid $\mathbf{HYB}_3^{i-1}$, except that the input of the ith honest party is replaced by $\chi(0)$. We now show that $\mathbf{HYB}_3^{i-1} \overset{c}{\approx} \mathbf{HYB}_3^i$ for $i \in [1, \ldots, H]$ which will let us conclude that $\mathbf{HYB}_3 \overset{c}{\approx} \mathbf{HYB}_4$. We fix an i and show that any $\mathcal{Z}_i$ that tells apart $\mathbf{HYB}_3^{i-1}$ and $\mathbf{HYB}_3^i$ can be turned into an attacker that can break semantic security of the SHE scheme.

Let $\mathcal{R}$ be the attacker that wants to break the semantic security of the SHE. The attacker participates in the semantic security game and receives from the challenger pk, ek and t keys corresponding to the corrupted parties for performing the distributed decryption. It then invokes $\mathcal{Z}_i$ (in head) to receive the inputs for the parties, say $(x_1, \ldots, x_n)$ and computes the function output y. The attacker prepares two messages, $\chi(0)$ and $\chi(x_i)$ for the challenger, the latter being received from $\mathcal{Z}_i$ as the input of P_i (namely x_i). In return, the attacker gets back $(\mathfrak{c}_{\mathbf{x}_i}, L)$ which either encrypts $\chi(0)$ or $\chi(x_i)$. Now the attacker computes encryptions of $\chi(0)$ for the first $(i-1)$ parties, for P_i the attacker uses $\mathfrak{c}_{\mathbf{x}_i}$ received from the challenger and for the remaining parties, the attacker computes encryptions of $\chi(x_{i+1}), \ldots, \chi(x_n)$. The attacker $\mathcal{R}$ then honestly evaluates the circuit on these encrypted inputs, ensuring all the similarities between $\mathbf{HYB}_3^{i-1}$ and $\mathbf{HYB}_3^i$. Namely, the the attacker performs the offline computation honestly and uses $(\mathfrak{c}_\mathbf{0}, 1)$ (an encryption of $\mathbf{0}$) instead of $(\mathfrak{c}_\mathbf{1}, 1)$ (as received from the $\mathcal{F}_{\text{SETUPGEN}}$). Moreover, while performing the randomization during the distributed decryption of the output wire ciphertext, the attacker uses an encryption of $\chi(y)$ as the randomizing ciphertext on the behalf of the honest party P_h (instead of an encryption of $\mathbf{0}$), so as to make the output wire ciphertext an encryption of $\chi(y)$. Furthermore, the attacker uses SHE.ShareSim to compute the decryption shares for the honest parties while performing the distributed decryption for the refresh gates and for the output wire. Note that the attacker will know the plaintext associated with the ciphertext to be decrypted (both for the refresh gates as well as for the output wire) while using SHE.ShareSim, even without knowing the actual circuit input of the party P_i (namely the plaintext associated with the challenge ciphertext $(\mathfrak{c}_{\mathbf{x}_i}, L)$) used for the circuit evaluation. This is because now $\mathfrak{c}_\mathbf{0}$ (instead of $c_\mathbf{1}$) is multiplied with the ciphertexts that are to be decrypted in the protocol and so the post-multiplication ciphertexts have associated plaintext $\mathbf{0}$, irrespective of the actual circuit inputs. This allows $\mathcal{R}$ to invoke SHE.ShareSim on a ciphertext for which it knows the associated plaintext even without knowing the inputs to the circuit. More specifically, for every refresh gate, $\mathcal{R}$ now knows the plaintext associated with the ciphertext to be decrypted, since it solely depends on the data created in offline computation which will be known to $\mathcal{R}$. On the other hand, for the output wire, $\mathcal{R}$ knows the plaintext associated with the ciphertext to be decrypted, since it is nothing but the circuit output $\chi(y)$. Finally at the end of the circuit evaluation as above, $\mathcal{R}$ outputs what $\mathcal{Z}_i$ outputs.

Now note that if the challenge ciphertext $(\mathfrak{c}_{\mathbf{x}_i}, L)$ is an encryption of $\chi(x_i)$, then $\mathcal{Z}_i$ is in $\mathbf{HYB}_3^{i-1}$, else it is in $\mathbf{HYB}_3^i$. The above reduction thus shows that $\mathcal{R}$ can distinguish between encryptions of $\chi(x_i)$ and $\chi(0)$ with the same probability with which $\mathcal{Z}_i$ can distinguish between $\mathbf{HYB}_3^{i-1}$ and $\mathbf{HYB}_3^i$. This implies that our claim is true.

$\mathbf{HYB}_4 \overset{s}{\approx} \mathbf{IDEAL}_{\mathcal{F}_f,\mathcal{S}_f^{\text{SH}},\mathcal{Z}}$**:** Follows from the inspection that the following steps have been performed in $\mathbf{HYB}_4$ as well $\text{IDEAL}_{\mathcal{F}_f,\mathcal{S}_f^{\text{SH}},\mathcal{Z}}$: (1) $\mathfrak{c}_1$ is replaced by $\mathfrak{c}_0$, (2) the inputs of the honest parties are replaced by $\chi(0)$s, (3) SHE.ShareSim is invoked to compute the decryption shares of the honest parties corresponding to all the refresh gates as well as in the output computation stage and (4) One of the honest party's randomizing ciphertext is an encryption of $\chi(y)$ instead of an encryption of $\mathbf{0}$.

Thus we have proved the following claim that in turn concludes the theorem.

Claim. $\text{IDEAL}_{\mathcal{F}_f,\mathcal{S}_f^{\text{SH}},\mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\Pi_f^{\text{SH}},\mathcal{A},\mathcal{Z}}$.

□

F Active Security

In this section we first discuss how to achieve a robust SHE.ShareCombine for our precise SHE scheme, then present a modified offline phase with linear communication overhead and then we go on to present the proof of Theorem 2.

F.1 Robust SHE.ShareCombine

Recall that in our concrete SHE scheme, the SHE.ShareCombine algorithm takes as input a set of shares obtained via Shamir Secret sharing over the ring R_{q_0}. From this observation it is clear, by the standard error correction properties of the Reed-Solomon codes (upon which the Shamir secret sharing is based), that one can obtain a robust SHE.ShareCombine algorithm immediately in the case of $t < n/3$.

All that remains is to present a robust SHE.ShareCombine for the case $t < n/2$. We present the protocol (note that SHE.ShareCombine will be now a protocol instead of a local algorithm as it may involve interaction among the parties) in Figure 7 that uses the dispute-control framework proposed in [4] and the fact that Reed-Solomon codes can detect up to $t < n/2$ errors. The protocol also invokes the ZK functionality for the relation $R_{sharedec}$ a limited number of times for the proof of correct (distributed) decryption, where $R_{sharedec}$ is given below.

$$R_{sharedec} = \{(((\mathfrak{c}, \mathfrak{l}), \bar{\mu}_i), \mathsf{dk}_i) \quad | \quad \bar{\mu}_i = \mathsf{SHE.ShareDec}_{\mathsf{dk}_i}(\mathfrak{c}, \mathfrak{l})\}$$

Unlike the functionality $\mathcal{F}_{\text{ZK}}^{R}$ defined in Figure 4 that treats all the parties in $\mathcal{P}$ as the verifiers, it is enough if the functionality for $R_{sharedec}$ is defined in a single prover and a single verifier settings. However we avoid elaborating more on this to keep simplicity.

Our robust SHE.ShareCombine realizes the following idea: For distributed decryption, as usual, every party sends the decryption shares to every other party. A party P_i on receiving the decryption shares first check whether all of them lie on a unique polynomial of degree at most t (namely error detection). If no error is detected then the secret can be safely reconstructed. However if some error is detected then P_i "complains" to the parties, asking them to prove the correctness of their respective decryption shares sent earlier; the parties respond back with ZK proofs by calling the $\mathcal{F}_{\text{ZK}}^{R_{sharedec}}$ functionality. Now P_i can "identify" the incorrect decryption share providers and ignore their shares in the future instances of distributed decryption. Each party P_i keeps a list $\mathcal{H}_i$ of the parties who it believes to be honest so far. Proper care has to be taken to ensure that the honest parties do not respond back "too many times" to the "false" complaints issued by the corrupted parties. This is resolved via keeping counters for the complaints. The idea is that an honest P_j will complain to an honest P_i at most t times and thus all the complaints from P_j after tth complaint clearly indicates that the complaint is false and P_j is corrupted. It is now easy to see that by using this trick, the *total* number of calls to $\mathcal{F}_{\text{ZK}}^{R_{sharedec}}$ in the MPC protocol will be $\mathcal{O}(n^3)$, which is independent of the circuit size; this is because a party may have to provide ZK proof to another party (by calling $\mathcal{F}_{\text{ZK}}^{R_{sharedec}}$) in at most t instances of distributed decryption. For large circuit sizes the extra communication cost to obtain a robust SHE.ShareCombine in the case $n/3 \leq t < n/2$ can be safely ignored.

SHE.ShareCombine

Each party P_i maintains its local copy $\mathcal{H}_i$ of a list all the parties which it currently assumed to be honest. Initially each $\mathcal{H}_i = \{P_1, \ldots, P_n\}$. Apart from this, every party P_i maintains n counters $\mathsf{cnt}_{i,1}, \ldots, \mathsf{cnt}_{i,n}$, where $\mathsf{cnt}_{i,j}$ is used to maintain a count of number of times an error message has been received from the party P_j; initially all these counters are set to 0. To execute an SHE.ShareCombine$((\mathfrak{c}, \mathfrak{l}), \{\bar{\mu}_j\}_{j \in \{1\ldots,n\}})$ operation, where $\bar{\mu}_j$ has been sent by P_j, party P_i performs the following steps:

- Ignore all $\bar{\mu}_j$ where $P_j \notin \mathcal{H}_i$. If the remaining $\bar{\mu}_j$s lie on a unique polynomial of degree at most t, then output the corresponding secret (namely the constant term of the polynomial). Otherwise, send a message $(\mathsf{sid}, i, \texttt{Error}_i, (\mathfrak{c}, \mathfrak{l}))$ to every party $P_j \in \mathcal{H}_i$.
- If an error message $(\mathsf{sid}, j, \texttt{Error}_j, (\mathfrak{c}, \mathfrak{l}))$ has been received from some $P_j \in \mathcal{H}_i$ then check whether $\mathsf{cnt}_{i,j} < t$. If $\mathsf{cnt}_{i,j} < t$, then call $\mathcal{F}_{\text{ZK}}^{R_{sharedec}}$ with the message $(\mathsf{sid}, i, j, (((\mathfrak{c}, \mathfrak{l}), \bar{\mu}_i), \mathsf{dk}_i))$ and set $\mathsf{cnt}_{i,j} := \mathsf{cnt}_{i,j} + 1$. Else if $\mathsf{cnt}_{i,j} \geq t$ then remove P_j from the list $\mathcal{H}_i$.
- If an error message $(\mathsf{sid}, i, \texttt{Error}_i, (\mathfrak{c}, \mathfrak{l}))$ has been sent in the first step, then execute the following: receive $(\mathsf{sid}, j, i, ((\mathfrak{c}, \mathfrak{l}), \bar{\mu}_j))$ from $\mathcal{F}_{\text{ZK}}^{R_{sharedec}}$ for every $P_j \in \mathcal{H}_i$. If for some $P_j \in \mathcal{H}_i$, the message $(\mathsf{sid}, j, i, \perp)$ is received from $\mathcal{F}_{\text{ZK}}^{R_{sharedec}}$ then remove P_j from $\mathcal{H}_i$. Using the $\bar{\mu}_j$s corresponding to the $P_j \in \mathcal{H}_i$, interpolate the polynomial of degree at most t, output its constant term as the secret.

Fig. 7. Robust SHE.ShareCombine For $t < n/2$

F.2 An Improved Offline Phase (sketch)

From the analysis in Section 6, we find that the online communication complexity of our protocol is Cost $= \mathcal{O}(n \cdot |\mathbb{G}_M|)$ (in the asymptotic sense). We now sketch that how we can modify our offline computation so that asymptotically the communication complexity of the offline phase is $\mathcal{O}(n \cdot \zeta)$, where $\zeta > \mathbb{G}_R$ is the number of random ciphertexts generate in the offline phase. We need the following three tools:

- Multi-valued Broadcast with $\mathcal{O}(n)$ Overhead [25]: This protocol allows a sender Sen $\in \{P_1, \ldots, P_n\}$ to send a message m of size ℓ "identically" to all the n parties (even if Sen is corrupted). The protocol can tolerate upto $t < n/2$ faults (even if the adversary is computationally unbounded) and has communication complexity $\mathcal{O}(n\ell)$ provided $\ell = \Omega(n^3)$.
- Randomness Extraction [31, 18]: Given a set of n encryptions of random values t of which may be known to the adversary, the randomness extraction algorithm based on superinvertible matrix [31] or Vandermonde matrix [18] allows the parties to (locally) compute encryptions of $(n - t)$ random values unknown to the adversary.
- Non-interactive Zero Knowledge Proofs: We require UC-secure instantiation of $\mathcal{F}_{\text{ZK}}^{R_{enc}}$, such that a party $P_i \in \{P_1, \ldots, P_n\}$ on computing encryptions of ℓ random values can publicly prove to anyone that it knows the associated plaintexts by "attaching" a proof of size $\mathcal{O}(\ell)$. Such proofs can be obtained, for example using the techniques of [2].

Now the offline phase protocol will proceed as follows: every party P_i computes encryptions of $\mathcal{L}$ random elements along with a NIZK proof that it knows the associated plaintexts where $\mathcal{L} = \frac{\zeta}{(n-t)}$. Party P_i then broadcasts the ciphertexts along with the proof by acting as a Sen and invoking the instance of a multi-valued broadcast protocol. The ciphertexts received from the different parties are then perceived as $\mathcal{L}$ batches of ciphertexts, where the lth batch consists of the lth ciphertext broadcasted by each party for $l \in [1, \ldots, \mathcal{L}]$. Finally, the randomness extraction algorithm on each batch of ciphertext to obtain $(n-t)$ random ciphertexts from each batch and in total $\mathcal{L} \cdot (n-t) = \zeta$ ciphertexts. Assuming $\mathcal{L} = \Omega(n^3)$, the total communication cost for the offline phase is now $\mathcal{O}(n \cdot \zeta)$: each instance of broadcast protocol has communication complexity $\mathcal{O}(n \cdot \mathcal{L}) = \mathcal{O}(\mathcal{L})$, as $(n-t) = \Theta(n)$. It is easy to see that the output ciphertexts are indeed random as there exists at least $(n-t)$ honest parties corresponding to each batch of ciphertexts. Note that we do not require any powerful (but somewhat complex) tools like player elimination, as used in the MPC protocol of [31] (whose communication complexity is also $\mathcal{O}(n \cdot \zeta)$).

F.3 Proof of Theorem 2

Since the robust $\mathsf{SHE.ShareCombine}$ works correctly even in the presence of t active corruptions, the correctness of our MPC protocol follows from the properties of $\mathcal{F}_{\mathsf{ZK}}^{R_{enc}}$ and $\mathcal{F}_{\mathsf{ZK}}^{R_{zeroenc}}$ by using the same arguments as used in Theorem 1. More specifically, the properties of $\mathcal{F}_{\mathsf{ZK}}^{R_{enc}}$ ensures that during the offline computation, each *corrupted* P_i knows the plaintext $\mathbf{m}_{ik}$ associated with the ciphertext $\mathfrak{c}_{\mathbf{m}_{ik}}$. Due to the same reason, each corrupted P_i knows the plaintext (namely the input) $\chi(x_i)$ associated with the ciphertext $\mathfrak{c}_{\mathbf{x}_i}$. Moreover, the property of $\mathcal{F}_{\mathsf{ZK}}^{R_{zeroenc}}$ ensures that each corrupted P_i has indeed contributed an encryption of $\mathbf{0}$ as a randomizing ciphertext during the distributed decryption of the output wire ciphertext. The homomorphic property of the SHE ensures that the addition and multiplication gates are evaluated correctly. We next argue that even the refresh gates are evaluated correctly. This follows because once the parties have access to the offline data, each refresh gate can be evaluated correctly if the parties are able to decrypt the corresponding masked ciphertext $\mathfrak{c}_{\mathbf{z}+\mathbf{m}}$. However since $\mathsf{SHE.ShareCombine}$ works even in the presence of t active corruptions, it follows that the parties can decrypt $\mathfrak{c}_{\mathbf{z}+\mathbf{m}}$. Due to the same reason, the parties will be able to decrypt the ciphertext associated with the output wire and hence can obtain the function output.

We next prove the security. Let $\mathcal{A}$ be a real-world active adversary up to t parties and let $T \subset \mathcal{P}$ denote the set of corrupted parties. We now present an ideal-world adversary (simulator) $\mathcal{S}_f^{\mathrm{MAL}}$ for $\mathcal{A}$ in Figure 8; for simplicity, we assume that an SHE with a robust, non-interactive $\mathsf{SHE.ShareCombine}$ (i.e. for $t < n/3$) has been used in the MPC protocol. The indistinguishability between the real and ideal world now follows mostly by the similar arguments given for semi-honest case (see the proof of Theorem 1 provided in Appendix E).

Simulator $\mathcal{S}_f^{\text{MAL}}$

Let SHE be a threshold L-levelled SHE scheme. The simulator plays the role of the honest parties and simulates each step of the protocol Π_f^{MAL} as follows. The communication of the $\mathcal{Z}$ with the adversary $\mathcal{A}$ is handled as follows: Every input value received by the simulator from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape. Likewise, every output value written by $\mathcal{A}$ on its output tape is copied to the simulator's output tape (to be read by the environment $\mathcal{Z}$). The simulator then does the following for session ID sid:

Offline Computation:

- On receiving the message (sid, i) to $\mathcal{F}_{\text{SETUPGEN}}$ from $\mathcal{A}$ for each $P_i \in T$, invoke $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}, \mathsf{dk}_1, \ldots, \mathsf{dk}_n) = \mathsf{SHE.KeyGen}(1^\kappa, n)$, compute $(\mathfrak{c}_{\mathbf{0}}, 1) = \mathsf{SHE.LowerLevel}_{\mathsf{ek}}(\mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{0}, \cdot), 1)$, and send $(\mathsf{sid}, \mathsf{pk}, \mathsf{ek}, \{\mathsf{dk}_i\}_{P_i \in T}, (\mathfrak{c}_{\mathbf{0}}, 1))$ to $\mathcal{A}$.
- For each party $P_j \notin T$ and $k \in [1, \ldots, \zeta]$, compute $(\mathfrak{c}_{\mathbf{m}_{jk}}, L) = \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{m}_{jk}, \cdot)$ for a randomly chosen $\mathbf{m}_{jk} \in \mathcal{M}$ and send $(\mathsf{sid}, j, (\mathfrak{c}_{\mathbf{m}_{jk}}, L))$ to $\mathcal{A}$ on the behalf of $\mathcal{F}_{\mathsf{ZK}}^{R_{enc}}$. For each $P_i \in T$ on receiving $(\mathsf{sid}, i, (\mathfrak{c}_{\mathbf{m}_{ik}}, L), (\mathbf{m}_{ik}, r_{ik}))$ as a message to $\mathcal{F}_{\mathsf{ZK}}^{R_{enc}}$ from $\mathcal{A}$ for $k \in [1, \ldots, \zeta]$, verify if $(\mathfrak{c}_{\mathbf{m}_{ik}}, L) \stackrel{?}{=} \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{m}_{ik}, r_{ik})$. If the verification fails for some $P_i \in T$ then send $(\mathsf{sid}, i, \perp)$ ζ times (corresponding to ζ ciphertexts) to $\mathcal{A}$ and set ζ publicly known level L encryptions of random values from $\mathcal{M}$ as $(\mathfrak{c}_{\mathbf{m}_{ik}}, L)$ for $k \in [1, \ldots, \zeta]$. Compute the kth ciphertext and the kth plaintext of the offline phase as in Π_f^{MAL}. The later can be computed by the simulator since it knows all the plaintexts.

Online Computation:

- **Input Stage**:
  - For every party $P_j \in \mathcal{P} \setminus T$, compute a random encryption $(\mathfrak{c}_{\mathbf{x}_j}, 1) = \mathsf{SHE.LowerLevel}_{\mathsf{ek}}(\mathsf{SHE.Enc}_{\mathsf{pk}}(\chi(0), \cdot), 1)$ and send $(\mathsf{sid}, j, (\mathfrak{c}_{\mathbf{x}_j}, 1))$ to $\mathcal{A}$ on the behalf of $\mathcal{F}_{\mathsf{ZK}}^{R_{enc}}$. For each $P_i \in T$ on receiving $(\mathsf{sid}, i, (\mathfrak{c}_{\mathbf{x}_i}, 1), (\chi(x_i), r_i))$ as a message to $\mathcal{F}_{\mathsf{ZK}}^{R_{enc}}$ from $\mathcal{A}$, verify $(\mathfrak{c}_{\mathbf{x}_i}, 1) \stackrel{?}{=} \mathsf{SHE.LowerLevel}_{\mathsf{ek}}(\mathsf{SHE.Enc}_{\mathsf{pk}}(\chi(x_i), r_i))$ and send $(\mathsf{sid}, i, \perp)$ to $\mathcal{A}$ if verification fails. Use publicly known ciphertext $(\mathfrak{c}_{\mathbf{x}_i}, 1)$ encrypting $x_i = \chi(0)$ on the behalf of any such P_i.
  - Send (sid, i, x_i) to $\mathcal{F}_f$ on the behalf of each $P_i \in T$ and receive the function output y.
- **Computation Stage**: The simulator acts in the same way as in $\mathcal{S}_f^{\text{SH}}$ except that whenever $\mathcal{A}$ sends the decryption shares corresponding to the parties in T during the evaluation of the refresh gates, the simulator ignores them; instead it computes the decryption shares by itself using the keys dk_i (for the distributed decryption) corresponding to $P_i \in T$ (the simulator knows dk_i for every $P_i \in T$ since it generated them by itself). These new decryption shares are then fed to $\mathsf{SHE.ShareSim}$ to obtain the simulated decryption shares corresponding to the honest parties, which the simulator then sends to $\mathcal{A}$ on behalf of the honest parties.
- **Output Stage**:
  - **Randomization**: Let $H = \mathcal{P} \setminus T$ be the set of honest parties and let P_h be some party in H. For every $P_j \in H \setminus \{P_h\}$ compute a random encryption $(\mathfrak{c}_j, L) = \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{0}, \cdot)$, while for $P_h \in H$ compute a random encryption $(\mathfrak{c}_h, L) = \mathsf{SHE.Enc}_{\mathsf{pk}}(\chi(y), \cdot)$. For every $P_j \in H$, send $(\mathsf{sid}, j, (\mathfrak{c}_j, L))$ to $\mathcal{A}$ on the behalf of $\mathcal{F}_{\mathsf{ZK}}^{R_{zeroenc}}$.
  - For each $P_i \in T$ on receiving $(\mathsf{sid}, i, (\mathfrak{c}_i, L), (\mathbf{0}, r_i'))$ as a message to $\mathcal{F}_{\mathsf{ZK}}^{R_{zeroenc}}$ from $\mathcal{A}$, verify if $(\mathfrak{c}_i, L) \stackrel{?}{=} \mathsf{SHE.Enc}_{\mathsf{pk}}(\mathbf{0}, r_i')$. If the verification fails for some $P_i \in T$ then send $(\mathsf{sid}, i, \perp)$ to $\mathcal{A}$ and consider a publicly known level L encryption of $\mathbf{0}$ as $(\mathfrak{c}_i, L)$ for such a P_i.
  - On receiving the decryption shares from $\mathcal{A}$ corresponding to the parties $P_i \in T$, the simulator ignores them and instead recomputes them using the dk_is and feed them to $\mathsf{SHE.ShareSim}$ to compute the simulated decryption shares for the honest parties. Finally it sends the simulated shares to $\mathcal{A}$ on behalf of the honest parties.

The simulator then outputs $\mathcal{A}$'s output.

Fig. 8. Simulator for the active adversary $\mathcal{A}$ corrupting t parties in the set $T \subset \mathcal{P}$.