Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE

Gilad Asharov* Abhishek Jain† Daniel Wichs‡

December 11, 2011

Abstract

Fully homomorphic encryption (FHE) provides a simple template for secure computation between *two* parties (Alice and Bob) where: (I) Alice encrypts her input under her key, (II) Bob homomorphically evaluates the desired function on Alice's ciphertext and his own input, and sends the encrypted output to Alice. Extending this approach to *multiple* parties raises the problem of which key to encrypt under; if all parties choose a key on their own, then homomorphic evaluation on ciphertexts under different keys will not be possible, and if a single party chooses the key for everyone then corrupting this party will break privacy for all.

In this work, we explore the option of using *threshold fully homomorphic encryption* (TFHE), allowing many parties to cooperatively generate a common public key whose secret key is shared/distributed among them. Moreover, the parties can cooperatively decrypt a ciphertext without learning anything but the plaintext. We show how to instantiate this approach efficiently using the recent FHE schemes of Brakerski et al. (FOCS '11, ITCS '12) based on the *learning with errors (LWE)* assumption. Our main tool is to exploit the property that such LWE-based encryption schemes are homomorphic over their *keys*.

Using TFHE, we construct multiparty computation (MPC) protocols secure against fully malicious settings, tolerating any number of corruptions, and providing security in the universal composability framework. Our schemes have several benefits over prior templates for MPC. **Interaction**: We get protocols with only 3 rounds of interaction in the common random string model, or 2 rounds with a reusable public-key infrastructure, improving on prior known results. **Communication**: The communication in our protocol is only proportional to the input and output size of the function being evaluated and independent of its circuit size. **Computation:** The only computation that depends on the size of the circuit being computed is a homomorphic evaluation over public ciphertexts. This computation can be performed by a single party or can be outsourced to an external server. **Novel Approach:** Prior approaches to MPC with a dishonest majority rely in part on some combination of the techniques of Yao (FOCS '86) and/or Goldreich, Micali and Wigderson (STOC '87). Our approach is fundamentally different and relies *only* on the homomorphic properties of LWE-based encryption.

Keywords: fully homomorphic encryption, threshold encryption, secure multiparty computation, round complexity, communication complexity.

*Bar-Ilan University. E-mail: asharog@cs.biu.ac.il. Supported by the European Research Council as part of the ERC project LAST. Research conducted while at IBM Research, T.J. Watson.

†UCLA. E-mail: abhishek@cs.ucla.edu. Research conducted while at IBM Research, T.J. Watson.

‡IBM Research, T. J. Watson. E-mail: wichs@cs.nyu.edu.

Contents

1 Introduction

Multiparty Computation. Secure multiparty computation (MPC) allows multiple participants to evaluate a common function over their inputs *privately*, without revealing the inputs to each other. This problem was initially studied by Yao [Yao82], who gave a protocol for the case of *two honest-but-curious parties* that follow the protocol specification honestly but wish to learn as much information as possible [Yao86]. The work of Goldreich, Micali and Wigderson [GMW87] extended this to *many fully malicious* parties that may arbitrarily deviate from the protocol specification. Since then, the problem of MPC has become a fundamental question in cryptography with much research activity. Interestingly, on a very high level, most prior results for general MPC can be seen as relying in some way on the original techniques of [Yao86, GMW87].

Fully Homomorphic Encryption. A very different approach to secure computation relies on *fully homomorphic encryption (FHE)*. An FHE scheme allows us to perform arbitrary computations on encrypted data without decrypting it. Although the idea of FHE goes back to Rivest et al. [RAD78], the first implementation is due to the recent breakthrough of Gentry [Gen09], and has now been followed with much exciting activity [vDGHV10, SV10, GH11, CMNT11, BV11b, BV11a, BGV12]. Using FHE, we immediately get an alternative approach to MPC in the case of two honest-but-curious parties (Alice and Bob): Alice encrypts her input under her own key and sends the ciphertext to Bob, who then evaluates the desired function homomorphically on Alice's ciphertext and his own input, sending (only) the final encrypted result back to Alice for decryption. This approach has several benefits over prior ones. Perhaps most importantly, the *communication complexity* of the protocol and Alice's *computation* are small and only proportional to Alice's input/output sizes, independent of the complexity of the function being evaluated. Moreover, the protocol consists of only *two rounds of interaction*, which is optimal (matching [Yao86]).[1]

MPC via Threshold FHE. Since FHE solves the secure computation problem for two honest-but-curious parties, it is natural to ask whether we can extend the above template to the general case of *many fully malicious* parties while maintaining its benefits. Indeed, there is a simple positive answer to this question (as pointed out in e.g. [Gen09]) by using a *threshold fully homomorphic encryption* (TFHE). This consists of a *key generation protocol* where the parties collaboratively agree on a common public key of an FHE scheme and each party also receives a share of the secret key. The parties can then encrypt their individual inputs under the common public key, evaluate the desired function homomorphically on the ciphertexts, and collaboratively execute a *decryption protocol* on the result to learn the output of the computation. Moreover, it is possible to convert any FHE scheme into TFHE by implementing the above key-generation and decryption protocols using general MPC compilers (e.g. [GMW87]). Although this approach already gives the communication/computation savings of FHE, it suffers from two main problems: (1) It does not preserve *round complexity* since generic implementations of the key-generation and decryption protocols will each require many rounds of interaction. (2) It uses the "heavy machinery" of generic MPC compilers and zero-knowledge proofs on top of FHE and is unlikely to yield practical solutions.

[1] Indeed, Yao's garbled circuits can be thought of as instantiating an FHE with long ciphertexts (see e.g. [GHV10]).

1.1 Our Results

In this work, we present an efficient threshold FHE scheme under the learning with errors (LWE) assumption, based on the FHE constructions of Brakerski, Gentry and Vaikuntanathan [BV11a, BGV12]. Our starting observation is that basic LWE-based encryption ([Reg05]) is *key homomorphic*, where summing up several public/secret key pairs (pk_i, sk_i) results in a new valid public/secret key pair $(pk^*, sk^*) = \sum_i (pk_i, sk_i)$. [2] Therefore, if each party broadcasts its own public-key pk_i, and we define the common public key as the sum $pk^* = \sum_i pk_i$, then each party already holds a share sk_i of the common secret key sk^*. Moreover, if each party decrypts a ciphertext c under pk^* with its individual share sk_i, then these partial decryptions can be summed up to recover message. This gives us simple key-generation and decryption protocols, consisting of one round each. Unfortunately, the above discussion is oversimplified and its implementation raises several challenges, which we are forced to overcome.

Smudging Distributions. The first challenge is that summing-up key pairs as above does not result in a correctly distributed fresh key pair, and summing up decryption shares may reveal more than just the plaintext. Nevertheless, we show the security of this basic approach when augmented with a technique we call *smudging*, in which parties add large noise during important operations so as to "smudge out" small differences in distributions.

Evaluation Keys. Perhaps our main challenge is that, in LWE-based FHE schemes, the public key must also contain additional information in the form of an *evaluation key*, which is needed to perform homomorphic operations on ciphertexts. Although the above key-homomorphic properties hold for the public *encryption keys* of the FHE, the evaluation keys have a more complex structure making it harder to combine them. Nevertheless, we show that it is possible to generate the evaluation keys in a threshold manner by having each party carefully release some extra information about its individual secret-key and then cleverly combining this information. Although this forces us to add an extra round to the key-generation protocol in order to generate the evaluation key, the parties can already encrypt their inputs after the first round. Therefore, we get MPC protocol consisting of only 3 broadcast rounds: (Round I) generate encryption key, (Round II) generate evaluation key & encrypt inputs, (Round III) perform homomorphic evaluation locally and decrypt the resulting ciphertext.

Security in the Fully Malicious Setting. Our basic TFHE protocol allows us to achieve MPC in the honest-but-curious model. To transform it to the fully malicious setting, we could use generic techniques consisting of: (1) coin-flipping for the random coins of each party, and (2) having each party prove at each step that it is following the protocol honestly (using the random coins determined by the coin-flip) by a zero-knowledge (ZK) proof of knowledge. Unfortunately, even if we were to use non-interactive zero knowledge (NIZK) in the common-random string (CRS) model for the proofs, the use of coin-flipping would add two extra rounds. Interestingly, we show that coin-flipping is *not* necessary. We do so by showing that our basic MPC protocol is already secure against a stronger class of attackers that we call *semi-malicious*: such attackers follow the

[2] This key-homomorphic property only holds if the keys are created with some common randomness (e.g. analogous to using a common generator for ElGamal encryption). Therefore, our protocols will assume some such common setup, which can be thought of as a common random string.

protocol honestly but with adaptively and adversarially chosen random coins in each round. We can now generically convert our MPC in the semi-malicious setting to a fully secure one using (UC) NIZKs [SCO+01] while *preserving the round complexity*. This gives the first 3 round protocol for general MPC in the CRS model (while achieving UC security for free).[3]

Efficient NIZKs in RO Model. Instantiating the above approach with general UC NIZKs proofs might already achieve asymptotic efficiency, but it has little hope of yielding practical protocols. Therefore, we also build efficient Σ-protocols [CDS94] for the necessary relations. We start with Σ-protocols for basic LWE-based languages (to the best of our knowledge, these are the first such protocols and may be of independent interest) and then show how to use these to construct Σ-protocols for the required relations using a series of *AND* and *OR* proofs. Lastly, we can compile the Σ-protocols into efficient UC NIZKs in the *random-oracle (RO)* model. Therefore, we can get an *efficient* and *simple* 3-round protocol for general MPC in the RO model.

1.2 Variants and Applications

Public-Key Infrastructure. Our approach also yields 2-round MPC in the *public-key infrastructure* (PKI) setting, by thinking of each party's original (Round I)-message as its public key and the randomness used to generate it as the secret key.[4] We can think of this PKI as being set-up honestly by a trusted party, or we can imagine that a trusted party only chooses a CRS and every party can then choose its own public key at will (possibly maliciously). This gives the first *two*-round MPC construction in the PKI setting, which is optimal (see [HLP11]). We note that the PKI can be *reused* for many MPC executions of arbitrary functions and arbitrary inputs.

Outsourced Computation. Next, we notice that each party's computation in the protocol is independent of the complexity of the evaluated circuit, *except* for the homomorphic evaluation over public ciphertexts. In the semi-honest setting, a single party or external entity can perform this evaluation and can tell the result to the rest of the parties at the expense of one extra round. In that case, the computation complexity of all other parties is independent of the evaluated circuit. In the fully-malicious setting, such outsourcing of computation would also require a short and efficiently verifiable proofs of correctness. These can be instantiated in 4 rounds [Kil92] or even made non-interactive in the random-oracle model [Mic00] or under appropriate non-black-box assumptions [BCCT12, GLR11, DFH11].

Fairness. We also give a variant of our protocol which achieves *fairness* in the case of an honest majority. In particular, an attacker cannot cause the protocol to abort, and honest parties always learn the output.

Application to Computing on the Web. We also note that our protocol is especially applicable in the setting of "secure computation on the web" [HLP11] where a single server or website coordinates the computation and parties "log in" at different times without coordination. Using our

[3] One downside is that our CRS is long and proportional to the number of parties. We leave it as an open problem to get rid of this inefficiency while maintaining round complexity.

[4] We rely on the fact that the (round I) message in our original protocol is already broadcast to everyone and does not depend on the parties' inputs. However, one downside of this approach is that the size of each public key is long and proportional to the total number of parties.

2-round MPC in the PKI model, we immediately get a solution to this problem with full security where the computation occurs in 2 stages (optimal) and each party "logs in" once per stage to post a message to the server. Note that the server does not do any processing on messages until the end of each stage. (Thus, the parties may, in fact ,"log in" *concurrently* in each stage.) Moreover, using the above ideas for "outsourcing" computation, only the computation of the server is proportional to the evaluated circuit. In contrast, the solution of [HLP11] only requires each party to log in once, but achieves a necessarily weaker notion of security. Furthermore, the server needs to perform non-trivial and *sequential* processing for each new party that logs in, and the computation of each party is large and proportional to the circuit size.

1.3 Related Work

Traditionally, in theory of cryptography, the round-complexity of protocols has been studied as an important measure of efficiency.[5] In the context of general MPC, starting from the original proposal of Yao [Yao86], there has been a rich line of work studying the round-complexity of secure two-party and multi-party computation protocols, both in the semi-honest and malicious settings.

The Semi-honest case: The original proposal of Yao [Yao86] in fact already gives a two-round two-party computation protocol. In the multi-party case, Beaver, Micali and Rogaway [BMR90] gave the first constant-round protocol, which is asymptotically optimal. An alternative approach using randomized polynomials was also given by [IK00, AIK05]. Although the concrete constants were not explicitly stated, they seem to require at least 4 rounds.

The Malicious case. In the fully malicious case, [BMR90] gave the first constant-round protocol for an *honest majority* ([DI05, DI06] later improved upon it to make only black-box use of the underlying primitives). In the two-party setting, Lindell [Lin01] gave the first constant-round protocol. Katz et al. [KOS03] gave the first constant-round multi-party protocol tolerating *dishonest majority*, by building on [BMR90, CLOS02] and using non black-box simulation techniques. Using recent results [Goy11, LP11], one can obtain similar result w.r.t. black-box simulation. Finally, we note that in the plain model, a lower bound of 5 rounds for the dishonest majority case was established by Katz and Ostrovsky [KO04]. Note that in the CRS model, we can compile any semi-honest protocol into one that is (UC) secure against malicious adversaries by using coin-tossing and (UC) NIZKs [SCO$^+$01], at the cost of adding *two extra rounds* for the coin flip. In contrast, our compiler (from semi-malicious to malicious) does not add any extra rounds.

Another line of work considers protocols split into a *"pre-processing"* stage (that is independent of the actual inputs and the circuit to be evaluated) and an "online" stage in which the inputs and the circuit become known. Recently, Choi et al [CEMY09] obtained a UC secure protocol in this model with a 2-round online stage. However, their pre-processing stage requires "expensive" computation of garbled circuits. Moreover, this pre-processing can only be used for a single future online computation and is *not* reusable. In contrast, our results give a 2-round UC-protocol in the PKI model, which we can think of as a very simple pre-processing that is only performed once and be reused for arbitrarily many computations in the future. The works of [FH96, JJ00, CDN01, DN03, BDOZ11] obtain multi-party protocols in the pre-processing model using the template of threshold *additively* homomorphic encryption, providing some of the most practically efficient MPC implementations. However, since the underlying encryption cannot handle multiplications, the

[5] In fact, recent implementations show that the latency of sending and receiving messages back and forth can be a dominating factor in running cryptographic protocols [MNPS04, BNP08].

round complexity in these works is large and linear in the depth of the circuit.

The work of Bendlin and Damgård [BD10] builds a threshold version of [Reg05] encryption based on LWE and ZK protocols for plaintext knowledge. Indeed the idea of using extra noise for "smudging" comes from that work, as do the main ideas behind our *decryption* protocol. We seem to avoid some of the main difficulties of [BD10] by analyzing the security of our threshold scheme directly within the application of MPC rather than attempting to realize ideal key-generation and decryption functionalities. However, we face a very different set of challenges in setting up the complicated evaluation key needed for FHE schemes based on LWE.

In a concurrent and independent work, Myers, Segi and shelat [MSS11] instantiate a threshold version of the FHE scheme of [vDGHV10] based on the "approximate-integer GCD" problem, and use it to build an explicit MPC protocol whose communication complexity is independent of the circuit size. Perhaps due to the amazing versatility and simplicity of LWE, our scheme enjoys several benefits over that of [MSS11], which only works in the setting of an honest majority and suffers from a large (constant) round-complexity. Most importantly, we believe that our protocol is significantly simpler to describe and understand.

2 Preliminaries

Throughout, we let κ denote the *security parameter* and $\mathsf{negl}(\kappa)$ denote a negligible function.

For integers n, q, we define $[n]_q$ to be the unique integer $v \in (-q/2, q/2]$ s.t. $n \equiv v \pmod q$. Let $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{Z}^n$ be a vector. We use the notation $\mathbf{x}[i] \stackrel{\text{def}}{=} x_i$ to denote the ith component scalar. To simplify the descriptions of our schemes, we also abuse notation and define $\mathbf{x}[0] \stackrel{\text{def}}{=} 1$. The ℓ_1-norm of $\mathbf{x}$ is defined as $\ell_1(\mathbf{x}) \stackrel{\text{def}}{=} \sum_{i=1}^{n} |x_i|$. For an integer N, we define $[N] = \{1, \ldots, N\}$.

For two distributions X, Y, over a finite domain Ω, the *statistical distance* between X and Y is defined by $\Delta(X, Y) \stackrel{\text{def}}{=} \frac{1}{2} \sum_{\omega \in \Omega} |X(\omega) - Y(\omega)|$. We write $\omega \leftarrow X$ to denote that ω is sampled at random according to distribution X. We write $\omega \stackrel{\$}{\leftarrow} \Omega$ to denote that it is sampled *uniformly at random* from the set Ω. For a distribution ensemble $\chi = \chi(\kappa)$ over the integers, and integers bounds $B = B(\kappa)$, we say that χ is *B-bounded* if $\Pr_{x \leftarrow \chi(\kappa)}[|x| > B(\kappa)] \leq \mathsf{negl}(\kappa)$.

We rely on the following lemma, which says that adding large noise "smudges out" any small values.

Lemma 2.1 (Smudging Lemma)**.** *Let $B_1 = B_1(\kappa)$, and $B_2 = B_2(\kappa)$ be positive integers and let $e_1 \in [-B_1, B_1]$ be a fixed integer. Let $e_2 \stackrel{\$}{\leftarrow} [-B_2, B_2]$ be chosen uniformly at random. Then the distribution of e_2 is* statistically indistinguishable *from that of $e_2 + e_1$ as long as $B_1/B_2 = \mathsf{negl}(\kappa)$.*

Proof: The statistical distance of the distributions is:

$$\frac{1}{2} \sum_{x=-(B_2+B_1)}^{(B_2+B_1)} |\Pr[e_2 = x] - \Pr[e_2 = x - e_1]| = \frac{1}{2} \left(\sum_{x=-(B_2+B_1)}^{-B_2} \frac{1}{B_2} + \sum_{x=B_2}^{B_2+B_1} \frac{1}{B_2} \right) = B_1/B_2$$

■

Learning With Errors. The *decisional learning with errors (*LWE*)* problem, introduced by Regev [Reg05], is defined as follows.

Definition 2.2 (LWE [Reg05]). *Let κ be the security parameter, $n = n(\kappa), q = q(\kappa)$ be integers and let $\chi = \chi(\kappa), \varphi = \varphi(\kappa)$ be distributions over $\mathbb{Z}$. The $\mathrm{LWE}_{n,q,\varphi,\chi}$ assumption says that no poly-time distinguisher can distinguish between the following distributions, given polynomially many samples:*
Distribution I. *Each sample $(\mathbf{a}_i, b_i) \xleftarrow{\$} \mathbb{Z}_q^{n+1}$ is chosen independently, uniformly at random.*
Distribution II. *Choose an initial value $\mathbf{s} \leftarrow \varphi^n$. Then choose each new sample $(\mathbf{a}_i, b_i)$ by taking $\mathbf{a}_i \xleftarrow{\$} \mathbb{Z}_q^n$, $e_i \leftarrow \chi$ and setting $b_i := \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$.*

The works of [Reg05, Pei09] show that the LWE problem is as hard as approximating short vector problems in lattices (for appropriate parameters) when χ is a Gaussian with "small" standard deviation and $\varphi = U(\mathbb{Z}_q)$ is the uniform distribution over $\mathbb{Z}_q$. The work of [ACPS09] shows that, when q is a prime power, then $\mathrm{LWE}_{n,q,\chi,\chi}$ is as hard as $\mathrm{LWE}_{n,q,U(\mathbb{Z}_q),\chi}$. Therefore, we can assume that the secret $\mathbf{s}$ of the LWE problem is "small" as well. It is also easy to see that, if q is odd, then $\mathrm{LWE}_{n,q,\varphi,(2\chi)}$ is as hard as $\mathrm{LWE}_{n,q,\varphi,\chi}$, where the distribution 2χ samples $e \leftarrow \chi$ and outputs $2e$.

One useful property of LWE shown in the following claim is that, given sufficiently many random LWE samples, the secret $\mathbf{s}$ is uniquely determined.

Claim 2.3 (Uniqueness of LWE Secrets). *Let $n, q, m > n\log(q) + \omega(\log(\kappa))$ and $B < q/8$ be integer parameters with q prime. Then with probability $1 - \mathsf{negl}(\kappa)$ over the choice of $A \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$ the following holds: for every $\mathbf{p} \in \mathbb{Z}_q^m$ there exists at most a single pair $(\mathbf{s}, \mathbf{e})$ with $\mathbf{s} \in \mathbb{Z}_q^n$ and $\mathbf{e} \in [-B, B]^m$ such that $\mathbf{p} = A\mathbf{s} + 2\mathbf{e}$.*

Proof: Assume that some $\mathbf{p}$ has two representations of the given form $\mathbf{p} = A\mathbf{s} + 2\mathbf{e} = A\mathbf{s}' + 2\mathbf{e}'$ with $\mathbf{s} \neq \mathbf{s}'$ and $\mathbf{e}, \mathbf{e}' \in [-B, B]^m$. Then $A(\mathbf{s} - \mathbf{s}')/2 = (\mathbf{e}' - \mathbf{e})$ and so there exists some non-zero $\mathbf{s}^* \in \mathbb{Z}_q^n$ such that $A\mathbf{s}^* \in [-2B, 2B]^m$. For any *fixed* $\mathbf{s}^* \neq 0$ the probability (over random A) that $A\mathbf{s}^* \in [-2B, 2B]^m$ is just $(4B/q)^m \leq 2^{-m}$. The claim follows by taking a union-bound over all q^n values $\mathbf{s}^* \in \mathbb{Z}_q^n$. ∎

3 Homomorphic Encryption from LWE

In this section, we give a brief description of the FHE schemes of [BV11a, BGV12].

3.1 Basic LWE-based Encryption

We start by describing a basic symmetric/public encryption scheme E, which is a variant of [Reg05] encryption scheme based on the LWE problem. This scheme serves as a building block for the more complex FHE schemes of [BV11a, BGV12] and of our threshold FHE scheme.

- $\mathsf{params} = (1^\kappa, q, m, n, \varphi, \chi)$ **:** The parameters of the scheme are an implicit input to all other algorithms, with: 1^κ is the security parameter, $q = q(k)$ is an odd modulus, $m = m(\kappa), n = n(\kappa)$ are the dimensions, and $\varphi = \varphi(\kappa)$, $\chi = \chi(\kappa)$ are distributions over $\mathbb{Z}_q$.
- $\mathsf{E.SymKeygen}(\mathsf{params})$ **– (symmetric key generation):** Choose a secret key $\mathbf{s} \leftarrow \varphi^n$.
- $\mathsf{E.PubKeygen}(\mathbf{s})$ **– (public key generation):** Choose $A \leftarrow \mathbb{Z}_q^{m \times n}$, $\mathbf{e} \leftarrow \chi^m$ and set $\mathbf{p} := A \cdot \mathbf{s} + 2 \cdot \mathbf{e}$. Output the public key $pk := (A, \mathbf{p})$ for the secret key $\mathbf{s}$.
- $\mathsf{E.SymEnc}_{\mathbf{s}}(\mu)$ **– (symmetric encryption):** To encrypt a message $\mu \in \{0, 1\}$, choose $\mathbf{a} \leftarrow \mathbb{Z}_q^n$, $e \leftarrow \chi$, and set $b \stackrel{\text{def}}{=} \langle \mathbf{a}, \mathbf{s} \rangle + 2 \cdot e + \mu$. Output the ciphertext $c = (\mathbf{a}, b)$.

- $\mathsf{E.PubEnc}_{pk}(\mu)$ **– (public key encryption):** To encrypt a message $\mu \in \{0,1\}$ under public key $pk = (A, \mathbf{p})$, choose $\mathbf{r} \leftarrow \{0,1\}^m$ and set $\mathbf{a} \stackrel{\text{def}}{=} \mathbf{r}^T \cdot A$, $b \stackrel{\text{def}}{=} \langle \mathbf{r}, \mathbf{p} \rangle + \mu$. Output $c = (\mathbf{a}, b)$. Note that: $b = \langle \mathbf{r}, \mathbf{p} \rangle + \mu = \langle \mathbf{r}, A \cdot \mathbf{s} + 2\mathbf{e} \rangle + \mu = \langle \mathbf{a}, \mathbf{s} \rangle + 2\langle \mathbf{r}, \mathbf{e} \rangle + \mu$. Therefore, ciphertexts have the same form as in the symmetric key scheme, but with an error distribution $2\langle \mathbf{r}, \mathbf{e} \rangle$.
- $\mathsf{E.Dec}_{\mathbf{s}}(c)$ **– (decryption):** To decrypt $c = (\mathbf{a}, b)$, output $[b - \langle \mathbf{a}, \mathbf{s} \rangle]_q \bmod 2$.

Under appropriate parameters and LWE assumption, the above scheme is semantically secure *with pseduorandom ciphertexts*, meaning that, given pk, a ciphertext of a chosen message is indistinguishable from a uniformly random ciphertext over the appropriate domain $\mathbb{Z}_q^{m+1}$

Theorem 3.1 (Variant of [Reg05])**.** *Assuming n, q, $m \geq (n+1)\log(q) + \omega(\log(\kappa))$ are integers with q odd, and that the* $\mathrm{LWE}_{n,q,\varphi,\chi}$ *assumption holds, the above* public key encryption scheme $(\mathsf{E.PubKeygen}, \mathsf{E.PubEnc}, \mathsf{E.Dec})$ *is semantically secure with* pseudorandom ciphertexts*.*

Proof: The view of the attacker consists of

$$pk = (A, \mathbf{p} = A\mathbf{s} + 2\mathbf{e}) \quad , \quad c = (\mathbf{r}^T A, \langle \mathbf{r}, \mathbf{p} \rangle + \mu) \quad : \quad A \xleftarrow{\$} \mathbb{Z}_q^{m \times n}, \mathbf{s} \leftarrow \varphi^n, \mathbf{r} \xleftarrow{\$} \{0,1\}^m, \mathbf{e} \leftarrow \chi^m$$

where $\mu \in \{0,1\}$. By the $\mathrm{LWE}_{n,q,\varphi,\chi}$ assumption, we can replace $\mathbf{p}$ with a uniformly random and independent value to get the indistinguishable distribution:

$$pk = (A, \mathbf{p}) \quad , \quad c = (\mathbf{r}^T A, \langle \mathbf{r}, \mathbf{p} \rangle + \mu) \quad : \quad A \xleftarrow{\$} \mathbb{Z}_q^{m \times n}, \mathbf{p} \xleftarrow{\$} \mathbb{Z}_q^m, \mathbf{r} \xleftarrow{\$} \{0,1\}^m$$

Since $h_{(A,\mathbf{p})}(\mathbf{x}) = (\mathbf{x}^T A, \langle \mathbf{x}, \mathbf{p} \rangle)$ is a universal hash function keyed by $(A, \mathbf{p})$, we can now apply the Leftover-Hash Lemma with $(A, \mathbf{p})$ as a "random seed" and $\mathbf{r}$ as the "source" to replace the "extracted randomness" $(\mathbf{r}^T A, \langle \mathbf{r}, \mathbf{p} \rangle)$ by uniform. Note that the min-entropy of $\mathbf{r}$ is $\mathbf{H}_\infty(\mathbf{r}) = m$ and the size of the extracted randomness is $(n+1)\log(q) \leq m - \omega(\log(\kappa))$. Therefore, we get the indistinguishable distribution:

$$pk = (A, \mathbf{p}) \quad , \quad c = (\mathbf{v}, w) \quad : \quad A \xleftarrow{\$} \mathbb{Z}_q^{m \times n}, \mathbf{p} \xleftarrow{\$} \mathbb{Z}_q^m, \mathbf{v} \xleftarrow{\$} \mathbb{Z}_q^n, w \xleftarrow{\$} \mathbb{Z}_q.$$

Lastly, we can apply $\mathrm{LWE}_{n,q,\varphi,\chi}$ once more to switch the public key back to an LWE tuple, giving us the indistinguishable distribution:

$$pk = (A, \mathbf{p} = A\mathbf{s} + 2\mathbf{e}) \quad , \quad c = (\mathbf{v}, w) \quad : \quad A \xleftarrow{\$} \mathbb{Z}_q^{m \times n}, \mathbf{s} \leftarrow \varphi^n, \mathbf{e} \leftarrow \chi^m, \mathbf{v} \xleftarrow{\$} \mathbb{Z}_q^n, w \xleftarrow{\$} \mathbb{Z}_q.$$

Therefore, an honestly generated encryption of the bit $\mu \in \{0,1\}$ is indistinguishable from a uniformly random ciphertext, even conditioned on pk. ■

Noise. We define the *noise* of a ciphertext $c = (\mathbf{a}, b)$ with respect to a key $\mathbf{s}$ and a modulus q to be $\mathsf{noise}_q(c, s) \stackrel{\text{def}}{=} [b - \langle \mathbf{a}, \mathbf{s} \rangle]_q$.[6] The parity of the noise is the decrypted message. In symmetric-key encryption, the noise is just $[2 \cdot e + \mu]_q$ where $e \leftarrow \chi$ is the randomness of the encryption algorithm. Decryption is therefore successful as long $|e| < q/4$. For public-key encryption, the noise is $[2\langle \mathbf{r}, \mathbf{e} \rangle + \mu]_q$ where $\mathbf{e} \leftarrow \varphi^m$ is the error in the public-key and $\mathbf{r} \xleftarrow{\$} \{0,1\}^m$ is the randomness of the encryption. Decryption is successful as long as $|\langle \mathbf{r}, \mathbf{e} \rangle| < q/4$.

[6] Recall, we define $[n]_q$ to be the unique integer $v \in (-q/2, q/2]$ s.t. $n \equiv v \pmod{q}$.

Approximate encryption. Although we defined symmetric/public key encryption for the message space $\mu \in \{0,1\}$, we can (syntactically) extend the same algorithms to any $\mu \in \mathbb{Z}_q$. Unfortunately, if μ is larger than a single bit, it will not be possible to decrypt μ correctly from the corresponding ciphertext. However, we can still think of this as an *approximate encryption* of μ, from which it is possible to recover the value $b - \langle \mathbf{a}, \mathbf{s} \rangle$ which is "close" to μ over $\mathbb{Z}_q$.

Fixing the coefficients. We use the notation $\mathsf{E.PubKeygen}(\mathbf{s}; A)$ to denote the execution of the key generation algorithm with fixed coefficients A (but still choosing a random error term $\mathbf{e}$). We use the notation $\mathsf{E.PubKeygen}(\mathbf{s}; A; \mathbf{e})$ to denote the execution of the key generation algorithm with the fixed coefficients A and the fixed error vector $\mathbf{e}$. In fact, this is a deterministic algorithm, and it determines the output uniquely. Similarly, we use the notation $\mathsf{E.SymEnc}_{\mathbf{s}}(\mu; \mathbf{a})$, $\mathsf{E.SymEnc}_{\mathbf{s}}(\mu; \mathbf{a}; e)$ to denote the analogue cases for the symmetric encryption algorithm.

3.2 Key-Homomorphic Properties of Basic Scheme

It is easy to see that the scheme E is additively homomorphic so that the sum of ciphertexts encrypts the sum of the plaintexts (at least as long as the noise is small enough and does not overflow). We now show it also satisfies several useful *key-homomorphic* properties, which make it particularly easy to convert into a threshold scheme. In particular, if we keep the coefficient vector $\mathbf{a}$ fixed, then summing just the b values of ciphertexts $c = (\mathbf{a}, b)$ gives an encryption of the sum of the plaintexts under the sum of the secret keys and the noise increases additively.

Claim 3.2. *Let $\mathbf{s}_1, \mathbf{s}_2$ be two secret keys, μ_1, μ_2 be two messages in $\{0,1\}$, $\mathbf{a}$ be some vector of coefficients and e_1, e_2 two noise values. Moreover, let $(\mathbf{a}, b_1) = \mathsf{E.SymEnc}_{\mathbf{s}_1}(\mu_1; \mathbf{a}; e_1)$ and $(\mathbf{a}, b_2) = \mathsf{E.SymEnc}_{\mathbf{s}_2}(\mu_2; \mathbf{a}; e_2)$. Then $(\mathbf{a}, b_1 + b_2) = \mathsf{E.SymEnc}_{\mathbf{s}_1+\mathbf{s}_2}(\mu_1 + \mu_2; \mathbf{a}; e_1 + e_2)$.*

Proof: Writing explicitly, we have $(\mathbf{a}, b_1) = \mathsf{E.SymEnc}_{\mathbf{s}_1}(\mu_1; \mathbf{a}; e_1) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s}_1 \rangle + 2e_1 + \mu_1)$ and $(\mathbf{a}, b_2) = \mathsf{E.SymEnc}_{\mathbf{s}_2}(\mu_2; \mathbf{a}; e_2) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s}_2 \rangle + 2e_2 + \mu_2)$, and so:

$$\begin{aligned}(\mathbf{a}, b_1 + b_2) &= (\mathbf{a}, \langle \mathbf{a}, \mathbf{s}_1 + \mathbf{s}_2 \rangle + 2(e_1 + e_2) + (\mu_1 + \mu_2)) \\ &= \mathsf{E.SymEnc}_{\mathbf{s}_1+\mathbf{s}_2}(\mu_1 + \mu_2; \mathbf{a}; e_1 + e_2).\end{aligned}$$

■

Also, if we keep A fixed, then the sum of two key pairs gives a new valid key pair.

Claim 3.3. *Let $\mathbf{s}_1, \mathbf{s}_2$ be two secret keys, let A be some matrix of coefficients and let $\mathbf{e}_1, \mathbf{e}_2$ be two error vectors. Moreover, let $(A, \mathbf{p}_1) = \mathsf{E.PubKeygen}(\mathbf{s}_1; A; \mathbf{e}_1)$ and $(A, \mathbf{p}_2) = \mathsf{E.PubKeygen}(\mathbf{s}_2; A; \mathbf{e}_2)$. Then, $(A, \mathbf{p}_1 + \mathbf{p}_2) = \mathsf{E.PubKeygen}(\mathbf{s}_1 + \mathbf{s}_2; A; \mathbf{e}_1 + \mathbf{e}_2)$.*

Proof: Explicitly, we have: $(A, \mathbf{p}_1) = \mathsf{E.PubKeygen}(\mathbf{s}_1; A; \mathbf{e}_1) = (A, A \cdot \mathbf{s}_1 + 2\mathbf{e}_1)$ and $(A, \mathbf{p}_2) = \mathsf{E.PubKeygen}(\mathbf{s}_2; A; \mathbf{e}_2) = (A, A \cdot \mathbf{s}_2 + 2\mathbf{e}_2)$. Then,

$$(A, \mathbf{p}_1 + \mathbf{p}_2) = A \cdot (\mathbf{s}_1 + \mathbf{s}_2) + 2 \cdot (\mathbf{e}_1 + \mathbf{e}_2) = \mathsf{E.PubKeygen}(\mathbf{s}_1 + \mathbf{s}_2; A; \mathbf{e}_1 + \mathbf{e}_2)$$

■

Security of Joint Keys. We show a useful security property of combining public keys. Assume that a public key $(A, \mathbf{p})$ is chosen honestly and an attacker can then *adaptively* choose some value $\mathbf{p}' = A\mathbf{s}' + 2e'$ for which it must *know* the corresponding $\mathbf{s}'$ and a "short" e'. Then the attacker cannot distinguish public-key encryptions under the combined key $(A, \mathbf{p} + \mathbf{p}')$ from uniformly random ones. Note that the combined key $\mathbf{p} + \mathbf{p}'$ may not be at all distributed like a correct public key. Indeed, we can only show that the above holds if the ciphertext under the combined key is "smudged" with additional large noise. We define the above property formally via the experiment $\mathsf{JoinKeys}_{\mathcal{A}}(\mathsf{params}, B_1, B_2)$ between an attacker $\mathcal{A}$ and a challenger, defined as follows:

1. Challenger chooses $\mathbf{s} \leftarrow \mathsf{E.SymKeygen}(\mathsf{params})$, and gives $(A, \mathbf{p}) \leftarrow \mathsf{E.PubKeygen}(\mathbf{s})$ to $\mathcal{A}$.
2. $\mathcal{A}$ adaptively chooses $\mathbf{p}', \mathbf{s}', \mathbf{e}'$ satisfying $\mathbf{p}' = A\mathbf{s}' + 2\mathbf{e}'$ and $\ell_1(\mathbf{e}') \leq B_1$. It also chooses $\mu \in \{0, 1\}$ and gives $(\mathbf{p}', \mathbf{s}', \mathbf{e}', \mu)$ to the challenger.
3. The challenger sets $pk^* := (A, \mathbf{p}^* = \mathbf{p} + \mathbf{p}')$. It chooses a random bit $\beta \xleftarrow{\$} \{0, 1\}$.
   - If $\beta = 0$ it chooses $\mathbf{a}^* \xleftarrow{\$} \mathbb{Z}_q^n$, $b^* \xleftarrow{\$} \mathbb{Z}_q$ uniformly at random.
   - Else it chooses $(\mathbf{a}^*, b) \leftarrow \mathsf{E.PubEnc}_{pk^*}(\mu)$, $e^* \xleftarrow{\$} [-B_2, B_2]$ and sets $b^* = b + 2e^*$.
4. $\mathcal{A}$ gets $(\mathbf{a}^*, b^*)$ and outputs a bit $\tilde{\beta}$.

The output of the experiment is 1 if $\tilde{\beta} = \beta$, and 0 otherwise.

Lemma 3.4. *Let* $\mathsf{params} = (1^\kappa, q, m, n, \varphi, \chi)$ *be a setting of parameters for which the give encryption scheme* E *has* pseudorandom ciphertexts. *Let* $B_1 = B_1(\kappa)$, $B_2 = B_2(\kappa)$ *be integers s.t.* $B_1/B_2 = \mathsf{negl}(\kappa)$. *Then, for any* PPT $\mathcal{A}$*:* $|\Pr[\mathsf{JoinKeys}_{\mathcal{A}}(\mathsf{params}, B_1, B_2) = 1] - \frac{1}{2}| = \mathsf{negl}(\kappa)$.

Proof: We build a reduction to the *pseudorandom ciphertexts* property of the scheme E. The reduction gets a challenge $pk = (A, \mathbf{p})$ and a "ciphertext" $(\mathbf{a}, b)$ which is either chosen as $\mathsf{E.PubEnc}_{pk}(0)$ ($\beta = 1$) or is uniformly random ($\beta = 0$). It gives $pk = (A, \mathbf{p})$ to $\mathcal{A}$ in the first round and receives back $\mathbf{p}', \mathbf{s}', \mathbf{e}'$ such that $\mathbf{p}' = A\mathbf{s}' + 2\mathbf{e}'$ and $\ell_1(\mathbf{e}') \leq B_1$. It then selects $e^* \xleftarrow{\$} [-B_2, B_2]$ and sets

$$\mathbf{a}^* = \mathbf{a}, \quad b^* = b + \langle \mathbf{a}, \mathbf{s}' \rangle + 2e^*.$$

Lastly, it sends $(\mathbf{a}^*, b^*)$ to $\mathcal{A}$ and outputs the bit $\tilde{\beta}$ produced by $\mathcal{A}$.

It is easy to see that, if $\beta = 0$, then $(\mathbf{a}^*, b^*)$ is just uniformly random. On the other hand, if $\beta = 1$, we can write $\mathbf{a} = \mathbf{r}A$, $b = \langle \mathbf{r}, \mathbf{p} \rangle$ for some $r \in \{0, 1\}^m$ therefore get:

$$\begin{aligned} b^* &= \langle \mathbf{r}, \mathbf{p} \rangle + \langle \mathbf{r}A, \mathbf{s}' \rangle + 2e^* = \langle \mathbf{r}, \mathbf{p} \rangle + (\langle \mathbf{r}, \mathbf{p}' \rangle - 2\langle \mathbf{r}, e' \rangle) + 2e^* = \langle \mathbf{r}, \mathbf{p}^* \rangle + 2(e^* - \langle \mathbf{r}, \mathbf{e}' \rangle) \\ &\overset{\mathrm{s}}{\equiv} \langle \mathbf{r}, \mathbf{p}^* \rangle + 2e^* \end{aligned}$$

where $\mathbf{p}^* = \mathbf{p} + \mathbf{p}'$. The second line above follows from Lemma 2.1 by noticing $|\langle \mathbf{r}, \mathbf{e}' \rangle| \leq \ell_1(\mathbf{e}') \leq B_1$. Therefore, the reduction acts indistinguishably from the real challenger with challenge bit β. Hence it breaks pseudorandomness of ciphertetxts with the same (up to negligible) advantage as $\mathcal{A}$. ■

3.3 Fully Homomorphic Encryption from LWE

In this section we present the construction of [BV11a, BGV12]. We start with the syntax of fully homomorphic encryption.

Definition. A *fully homomorphic (public–key) encryption* (FHE) scheme is a quadruple of PPT algorithms $\mathsf{FHE} = (\mathsf{FHE.Keygen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ defined as follows.

- $\mathbf{FHE.Keygen}(1^\kappa) \to (\boldsymbol{pk}, \boldsymbol{evk}, \boldsymbol{sk})$: Outputs a public encryption key pk, a public evaluation key evk and a secret decryption key sk.
- $\mathbf{FHE.Enc}_{\boldsymbol{pk}}(\boldsymbol{\mu}) \to \boldsymbol{c}$: Encrypts a bit $\mu \in \{0,1\}$ under public key pk. Outputs ciphertext c.
- $\mathbf{FHE.Dec}_{\boldsymbol{sk}}(\boldsymbol{c}) \to \boldsymbol{\mu^*}$: Decrypts ciphertext c using sk. Outputs plaintext bit $\mu^* \in \{0,1\}$.
- $\mathbf{FHE.Eval}_{\boldsymbol{evk}}(\boldsymbol{f}, \boldsymbol{c_1}, \ldots, \boldsymbol{c_\ell}) \to \boldsymbol{c_f}$: The *homomorphic evaluation algorithm* is a *deterministic* poly-time algorithm that takes the evaluation key evk, a boolean circuit $f : \{0,1\}^\ell \to \{0,1\}$, and a set of ℓ ciphertexts $c_1, \ldots, c_\ell$. It outputs the result ciphertext c_f.

We say that an FHE scheme is secure if it satisfies the standard notion of *semantic security* for public-key encryption, where we consider the evaluation key evk as a part of the public key. Next we define what it means for a scheme to be homomorphic.

Definition 3.5 ((Leveled) Fully Homomorphic Encryption.)**.** *An encryption scheme is* fully homomorphic *if for any boolean circuit* $f : \{0,1\}^\ell \to \{0,1\}$ *and respective inputs* $\mu_1, \ldots, \mu_\ell \in \{0,1\}$*, it holds that:*

$$\Pr\left[\mathsf{FHE.Dec}(\mathsf{FHE.Eval}_{evk}\left(f, c_1, \ldots, c_\ell\right) \neq f(\mu_1, \ldots, \mu_\ell)\right] \leq \mathsf{negl}(\kappa)$$

where $(pk, evk, sk) \leftarrow \mathsf{FHE.Keygen}(1^\kappa)$ *and* $c_i \leftarrow \mathsf{FHE.Enc}_{pk}(\mu_i)$*. We say that the scheme is a* leveled fully homomorphic *if the* $\mathsf{FHE.Keygen}$ *algorithm gets an additional (arbitrary) input* 1^D *and the above only holds for circuits* f *consisting of at most* D *multiplicative levels.*

Construction. We give an overview of the FHE construction of [BV11a, BGV12]. See Appendix B for a full description. The construction begins with the basic encryption scheme E which is already additively homomorphic. We associate ciphertexts $c = (\mathbf{a}, b)$ under E with symbolic polynomials

$$\phi_c(\mathbf{x}) \stackrel{\text{def}}{=} b - \langle \mathbf{a}, \mathbf{x} \rangle \quad : \quad \text{an } n\text{-variable degree-1 polynomial over } \mathbf{x}.$$

so that $\mathsf{Dec}_{\mathbf{s}}(c) = [\phi_c(\mathbf{s})]_q \bmod 2$. If c_1, c_2 encrypt bits μ_1, μ_2 under a secret key $\mathbf{s}$, we can define the polynomial $\phi_{mult}(\mathbf{x}) \stackrel{\text{def}}{=} \phi_{c_1}(\mathbf{x}) \cdot \phi_{c_2}(\mathbf{x})$. This already "encrypts" $\mu_1 \cdot \mu_2$ in the sense that $[\phi_{mult}(\mathbf{s})]_q = \mu_1 \cdot \mu_2 + 2e^*$ where e^* is "small". Unfortunately, ϕ_{mult} is a degree-2 polynomial and hence its description is much larger than that of the original ciphertexts c_1, c_2.

The main challenge is to *re-linearize* the polynomial ϕ_{mult} to convert it into a degree-1 polynomial ϕ'_{mult} which still encrypts $\mu_1 \cdot \mu_2$. Such re-linearization is possible with two caveats: (1) The polynomial ϕ'_{mult} encrypts $\mu_1 \cdot \mu_2$ under a *new* key $\mathbf{t}$. (2) We need to know additional ciphertexts $\psi_{i,j,\tau}$ that (approximately) encrypt information about the key $\mathbf{s}$ under a new key $\mathbf{t}$ as follows:[7]

$$\{\psi_{i,j,\tau} \leftarrow \mathsf{E.SymEnc}_{\mathbf{t}}(\ 2^\tau \cdot \mathbf{s}[i] \cdot \mathbf{s}[j]) \quad : \quad i \in [n], j \in [n] \cup \{0\}, \tau \in \lfloor \{0, \ldots, \log(q)\rfloor \}\}\,.$$

See Appendix B for the details of this re-linearization procedure. The above ideas give us *leveled homomorphic encryption scheme* for circuits with D multiplicative levels simply by choosing $D+1$

[7] Recall, we define $\mathbf{s}[0] \stackrel{\text{def}}{=} 1$.

secret keys $\mathbf{s}_0, \ldots, \mathbf{s}_D$ and publishing the ciphertexts $\{\psi_{d,i,j,\tau}\}$ which encrypt the required information about the level-d secret $\mathbf{s}_d$ under level-$(d+1)$ secret $\mathbf{s}_{d+1}$. The public key of the scheme is $pk \leftarrow \mathsf{E.PubKeygen}(\mathbf{s}_0)$, corresponding to the level-0 secret key $\mathbf{s}_0$. The ciphertexts will have an associated level number, which is initially 0. Each time we multiply two ciphertexts with a common level d, we need to perform re-linearization which increases the level to $d+1$. Using the secret key $\mathbf{s}_D$, we can then decrypt at the top level.

In the above discussion, we left out the crucial question of *noise*, which grows exponentially with the number of multiplications. Indeed, the above template only allows us to evaluate some logarithmic number of levels before the noise gets too large. The work of [BGV12] gives a beautifully simply noise-reduction technique called "modulus reduction". This technique uses progressively smaller moduli q_d for each level d and simply "rescales" the ciphertext to the smaller modulus to reduce its noise level. See Appendix B for further details. As an end result, we get a *leveled* FHE scheme, allowing us to evaluate circuits containing at most D multiplicative levels, where D is an arbitrary polynomial, used as a parameter for $\mathsf{FHE.Keygen}$. To get an FHE scheme where key generation does not depend on the number of levels, we can apply the bootstrapping technique of [Gen09], at the expense of having to make an additional "circular security assumption".

4 Threshold Fully Homomorphic Encryption

Syntax. A threshold fully homomorphic encryption scheme (TFHE) is basically a homomorphic encryption scheme, with the difference that the Keygen and Dec are now N-party *protocols* instead of algorithms. We will consider protocols defined in terms of some common setup.

- **$\mathsf{TFHE.Keygen}(\mathsf{setup})$ – (key generation protocol):** Initially each party holds setup. At the conclusion of the protocol, each party P_k, for $k \in [N]$ outputs a common public-key pk, a common public evaluation key evk, and a private *share* sk_k of the implicitly defined secret key sk.
- **$\mathsf{TFHE.Dec}_{sk_1,\ldots,sk_n}(c)$ – (decryption protocol):** Initially, each party P_k holds a common ciphertext c and its private share of the secret key sk_k. At the end of the protocol each party receives the decrypted plaintext μ.
- **$\mathsf{TFHE.Enc}_{pk}(\mu), \mathsf{TFHE.Eval}_{pk}(f, c_1, \ldots, c_\ell)$**: Encryption and evaluation are *non-interactive algorithms* with the same syntax as in FHE.

We do *not* define the security of TFHE on its own. Indeed, requiring that the above protocols securely realize some ideal key-generation and decryption functionalities is unnecessarily restrictive. Instead, we will show that our TFHE scheme is secure directly in the context of our implementation of general MPC in section 5.

4.1 Construction of TFHE

We now give our construction of TFHE, which can be thought of as a threshold version of the [BGV12] FHE scheme. The main difficulty is to generate the evaluation key in a threshold manner, by having each party carefully release some extra information about its key-shares. Another important component of our construction is to require parties to add some additional *smudging* noise during sensitive operations, which will be crucial when analyzing security.

Common Setup. All parties share a common setup consisting of:

- $\mathsf{params} = \Big(\{\mathsf{params}_d = (1^\kappa, q_d, m, n, \varphi, \chi)\}_{0\le d\le D}\ ,\ B_\varphi, B_\chi, B^{eval}_{smdg}, B^{enc}_{smdg}, B^{dec}_{smdg}\Big)$, where
  - params_d are parameters for the encryption scheme E with differing moduli q_d.
  - $B_\varphi, B_\chi \in \mathbb{Z}$ are bounds s.t. φ is B_φ-bounded and χ is B_χ-bounded.
  - $B^{eval}_{smdg}, B^{enc}_{smdg}, B^{dec}_{smdg} \in \mathbb{Z}$ are bounds for extra "smudging" noise.
- Randomly chosen common values (i.e. a *common random string* or CRS):

$$\{A_d \xleftarrow{\$} \mathbb{Z}_{q_d}^{m\times n}\}_{d\in\{0,\ldots,D\}} \quad , \quad \left\{\mathbf{a}^k_{d,i,\tau} \xleftarrow{\$} \mathbb{Z}^n_{q_d} \quad : \quad \begin{array}{l} k\in[N], i\in[n] \\ d\in[D], \tau\in\{0,\ldots,\lfloor\log(q_d)\rfloor\}\end{array}\right\}.$$

Convention. Whenever the protocol specifies that a party is to sample $x \leftarrow \varphi$ (resp. $x \leftarrow \chi$), we assume that it checks that $|x| \le B_\varphi$ (resp. $|x| \le B_\chi$) and re-samples if this is not the case (which happens with negligible probability). This will be crucial when analyzing semi-malicious security, since it implies that a semi-malicious attacker needs to stay within these bounds.

TFHE.Keygen(setup). This is a two-round protocol between N parties.

- **Round 1:**
  1. Each party P_k invokes the key generation algorithm of the basic scheme E for each level $d \in \{0,\ldots,D\}$ to get $\mathbf{s}^k_d \leftarrow \mathsf{E.SymKeygen}(\mathsf{params}_d)$ and
  $$(A_d, \mathbf{p}^k_d) \leftarrow \mathsf{E.PubKeygen}(\mathbf{s}^k_d\ ;\ A_d)$$
  so that $\mathbf{p}^k_d = A_d\cdot\mathbf{s}^k_d + 2\cdot\mathbf{e}^k_d$ for some noise $\mathbf{e}^k_d$. We can think of the values $\mathbf{s}^k_d$ as *individual secret keys* and $\mathbf{p}^k_d$ as *individual encryption keys* of each party P_k.
  2. For every $d\in[D]$, $i\in[n]$, $\tau\in\{0,\ldots,\lfloor\log q\rfloor\}$, the party P_k computes:
  $$\left(\mathbf{a}^k_{d,i,\tau}, b^{k,k}_{d,i,\tau}\right) \leftarrow \mathsf{E.SymEnc}_{\mathbf{s}^k_d}\left(\ 2^\tau\cdot\mathbf{s}^k_{d-1}[i]\ ;\ \mathbf{a}^k_{d,i,\tau}\ \right)$$
  so that $b^{k,k}_{d,i,\tau} = \langle\mathbf{a}^k_{d,i,\tau}, \mathbf{s}^k_d\rangle + 2e^{k,k}_{d,i,\tau} + 2^\tau\cdot\mathbf{s}^k_{d-1}[i]$ for some small noise $e^{k,k}_{d,i,\tau}$. In addition, for every d, i, τ as above and $\ell\in[N]\setminus\{k\}$, the party P_k computes "encryptions of 0":
  $$\left(\mathbf{a}^\ell_{d,i,\tau}, b^{\ell,k}_{d,i,\tau}\right) \leftarrow \mathsf{E.SymEnc}_{\mathbf{s}^k_d}(0\ ;\ \mathbf{a}^\ell_{d,i,\tau})$$
  so that $b^{\ell,k}_{d,i,\tau} = \langle\mathbf{a}^\ell_{d,i,\tau}, \mathbf{s}^k_d\rangle + 2e^{\ell,k}_{d,i,\tau}$ for some noise $e^{\ell,k}_{d,i,\tau}$. The values $\{b^{\ell,k}_{d,i,\tau}\}$ will be used to create the evaluation key.
  3. Each party P_k broadcasts the values $\quad \left\{\mathbf{p}^k_d\right\}_d\ , \left\{b^{\ell,k}_{d,i,\tau}\right\}_{\ell,d,i,\tau}$.
- **End of Round 1:** At the end of round 1, we can define the following values.
  1. For every $d\in\{0,\ldots,D\}$, define: $\mathbf{p}^*_d := \sum_{\ell=1}^N \mathbf{p}^\ell_d$. Let $pk := (A_0, \mathbf{p}^*_0)$ be the common public encryption key of the TFHE scheme.

     Notice that, if all parties follow the protocol then $(A_d, \mathbf{p}^*_d) = \mathsf{E.PubKeygen}(\mathbf{s}^*_d; A_d; \mathbf{e}^*_d)$. where $\mathbf{s}^*_d := \sum_{\ell=1}^N \mathbf{s}^\ell_d$, $\mathbf{e}^*_d := \sum_{\ell=1}^N \mathbf{e}^\ell_d$. We can think of these values as the "combined public keys" for each level d.

2. For every $\ell \in [N], d \in [D]$, and all i, τ define $\beta^{\ell}_{d,i,\tau} := \sum_{k=1}^{N} b^{\ell,k}_{d,i,\tau}$.

   Notice that, if all parties follow the protocol then:

$$(\mathbf{a}^{\ell}_{d,i,\tau}, \beta^{\ell}_{d,i,\tau}) = \mathsf{E.SymEnc}_{\mathbf{s}^*_d}(\ 2^{\tau} \cdot \mathbf{s}^{\ell}_{d-1}[i]\ ;\ \mathbf{a}^{\ell}_{d,i,\tau}\ ;\ e\ ) \quad \text{where } e = \sum_{k=1}^{N} e^{\ell,k}_{d,i,\tau}$$

   These "approximate encryptions" are already encrypted under the correct combined secret key $\mathbf{s}^*_d$ of level d. However, the "plaintexts" still only correspond to the *individual secret keys* $\mathbf{s}^{\ell}_{d-1}$ at level $d-1$, instead of the desired combined key $\mathbf{s}^*_{d-1}$. We fix this in the next round.

- **Round 2:**

  1. Each party P_k does the following. For all $\ell \in [N], d \in [D]$, $i, j \in [n]$, $\tau \in \{0, \ldots, \lfloor \log q \rfloor\}$: sample $(\mathbf{v}^{\ell,k}_{d,i,j,\tau}, w^{\ell,k}_{d,i,j,\tau}) \leftarrow \mathsf{E.PubEnc}_{p^*_d}(0)$ and $e \xleftarrow{\$} [-B^{eval}_{smdg}, B^{eval}_{smdg}]$. Set:

$$(\boldsymbol{\alpha}^{\ell,k}_{d,i,j,\tau}, \beta^{\ell,k}_{d,i,j,\tau}) \quad := \quad \mathbf{s}^{k}_{d-1}[j] \cdot (\mathbf{a}^{\ell}_{d,i,\tau}, \beta^{\ell}_{d,i,\tau}) + (\mathbf{v}^{\ell,k}_{d,i,j,\tau}, w^{\ell,k}_{d,i,j,\tau} + 2e)$$

     Note that, if all parties follow the protocol, then the original tuple $(\mathbf{a}^{\ell}_{d,i,\tau}, \beta^{\ell}_{d,i,\tau})$ approximately encrypts the value $2^{\tau}\mathbf{s}^{\ell}_{d-1}[i]$. The above operation has party P_k "multiply in" its component $\mathbf{s}^{k}_{d-1}[j]$ (and re-randomizing via a public encryption of 0) so that the final tuple $(\boldsymbol{\alpha}^{\ell,k}_{d,i,j,\tau}, \beta^{\ell,k}_{d,i,j,\tau})$ approximately encrypts $2^{\tau} \cdot \mathbf{s}^{\ell}_{d-1}[i] \cdot \mathbf{s}^{k}_{d-1}[j]$.

  2. Each party P_k broadcasts the ciphertexts $\left\{(\boldsymbol{\alpha}^{\ell,k}_{d,i,j,\tau}, \beta^{\ell,k}_{d,i,j,\tau})\right\}_{d,i,j,\tau,\ell}$.

- **End of Round 2:** At the end of round 2, we can define the following values.

  1. We define the *combined evaluation key* components for all $d \in [D]$ and all $i \in [n], j \in [n] \cup \{0\}, \tau$ as:

$$\psi_{d,i,j,\tau} \quad := \quad \begin{cases} \sum_{\ell=1}^{N} \sum_{k=1}^{N} (\boldsymbol{\alpha}^{\ell,k}_{d,i,j,\tau}, \beta^{\ell,k}_{d,i,j,\tau}) & j \neq 0 \\ \sum_{\ell=1}^{N} (\mathbf{a}^{\ell}_{d,i,\tau}, \beta^{\ell}_{d,i,\tau}) & j = 0 \end{cases}$$

     Note that, if all parties follow the protocol, then

$$\psi_{d,i,j,\tau} \quad = \quad \mathsf{E.SymEnc}_{\mathbf{s}^*_d}(2^{\tau} \cdot \mathbf{s}^*_{d-1}[i] \cdot \mathbf{s}^*_{d-1}[j])$$

     where $\mathbf{s}^*_d := \sum_{\ell=1}^{N} \mathbf{s}^{\ell}_d$ is the combined secret key and all "errors" are "sufficiently small".

- **Outputs:**

  1. *Public Evaluation key:* Output $evk = \{\psi_{d,i,j,\tau}\}_{d,i,j,\tau}$ as the evaluation key.
  2. *Public Encryption key:* Output $pk = (A_0, \mathbf{p}^*_0)$ as the public key.
  3. *Share of secret key:* Each party P_k has a secret-key share $\mathbf{s}^k_D$.

$\mathsf{TFHE.Enc}_{pk}(\mu)$: Once the *first round* of the key-generation protocol is concluded, the public key $pk = (A, \mathbf{p}^*_0)$ is well defined. At this point anybody can encrypt as follows. Choose $(\mathbf{v}, w) \leftarrow \mathsf{E.Enc}_{pk}(\mu)$ using the basic scheme E with the parameters $\mathsf{params}_0 = (1^{\kappa}, q_0, m, n, \varphi, \chi)$. Choose additional "smudging noise" $e \xleftarrow{\$} [-B^{enc}_{smdg}, B^{enc}_{smdg}]$ and output the ciphertext $c = ((\mathbf{v}, w + 2e), 0)$ with associated "level" 0.

$\mathsf{TFHE.Eval}_{evk}(f, c_1, \ldots, c_t)$: Once the *second round* of the key-generation protocol is concluded, the evaluation key evk is defined. The evaluation algorithm is exactly the same as that of the underlying scheme FHE of [BGV12]. See Appendix B for details.

The Decryption Protocol: $\mathsf{TFHE.Dec}(c)$. This is a one-round protocol between N parties. Initally all parties hold a common ciphertext $c = (\mathbf{v}, w, D)$ with associated "level" D.[8] Moreover, each party P_k holds its share $\mathbf{s}_D^k$ for the joint secret key $\mathbf{s}_D^* = \sum_{k=1}^N \mathbf{s}_D^k$. At the end all parties get the decrypted bit μ.

- Each party P_k broadcasts $w^k = \langle \mathbf{v}, \mathbf{s}_D^k \rangle + 2 \cdot e_k$ for some noise $e_k \xleftarrow{\$} [-B_{smdg}^{dec}, B_{smdg}^{dec}]$.
- Given $w^1, \ldots, w^N$, each P_k computes the output bit: $\mu = [w - \sum_{i=1}^N w^i]_{q_D} \mod 2$.

Correctness. In Appendix C.1 we show that the above protocol satisfies *correctness* (for appropriate parameters) in the following sense. Assume that: (1) the key-generation protocol is executed honestly, (2) data $\mu \in \{0,1\}^*$ is encrypted under the public encryption key bit-wise and some homomorphic evaluation corresponding to a function f is performed on the ciphertexts, (3) the decryption protocol is executed honestly on the resulting ciphertexts. Then the output of the decryption protocol is $f(\mu)$. In fact, we show a stronger statement that the above holds even if the protocols and encryptions aren't executed with honestly and independently chosen randomness for each party, but we are only guaranteed that the noise levels in each step are sufficiently bounded.

5 Secure MPC via TFHE

PROTOCOL 5.1 (π_f: Secure MPC Protocol for f.)**.**

Let $f : (\{0,1\}^{\ell_{in}})^N \to \{0,1\}^{\ell_{out}}$ be a function computed by a circuit of multiplicative depth D.

Input: Each party P_k has input $\mathbf{x}_k \in \{0,1\}^{\ell_{in}}$. The parties share some common setup for our D-level TFHE scheme, where D is the number of levels for f.

The Protocol:

Round I. The parties execute the *first* round of the TFHE.Keygen protocol. At the end of this round, each party P_k holds the common public key pk and a secret-key share sk_k.

Round II. The parties execute the *second* round of the TFHE.Keygen protocol. Concurrently, each party P_k also encrypts its input $\mathbf{x}_k$ bit-by-bit under the common public key pk and broadcasts the corresponding ciphertexts $\{ c_{k,i} \leftarrow \mathsf{TFHE.Enc}_{pk}(\mathbf{x}_k[i]) \}_{i \in \{1,\ldots,\ell_{in}\}}$.

At the end of this round, each party can locally compute the common evaluation key evk, and homomorphically evaluate the function f to get the output ciphertexts $\{ c_j^* := \mathsf{Eval}_{evk}(f_j; \{c_{k,i}\}) \}_{j \in \{1,\ldots,\ell_{out}\}}$ where f_j is the boolean function for the jth bit of f.

Round III. The parties execute the decryption protocol TFHE.Dec on each of the output ciphertexts $\{c_j^*\}$ concurrently. At the end of this invocation, each party learns each of the bits of the underlying plaintext $y = f(x_1, \ldots, x_N)$, which it sets as its output.

[8] We can assume w.l.o.g. that the ciphertext to be decrypted has level exactly D (rather than at most D) since we can always deterministically upgrade the ciphertext level to D. See Appendix B.

We now present a protocol for general MPC, using the threshold fully homomorphic scheme TFHE from the previous section. The protocol, given in Figure 5.1, realizes general multiparty computation for any polynomial-time deterministic functions f which produces a common output for all parties. It does so with respect to a static *semi-malicious attackers* corrupting any $t \leq N$ parties. We extend this result to deal with probabilistic functions with individual outputs in Section 6 and to fully malicious attackers in Section E. We state our main theorem without concrete parameters. The proof appears in Appendix C.2, where we also discuss the settings of the parameters for our protocol and the corresponding LWE assumption required for security.

Theorem 5.2. *Let f be any deterministic poly-time function with N inputs and single output. Let* params *satisfy the constraints given in Appendix C.2, and assume that the corresponding LWE assumption holds. Then the protocol π_f securely UC-realizes the ideal functionality $\mathcal{F}_f$ in the presence of a static* semi-malicious *adversary (Definition A.2) corrupting any $t \leq N$ parties.*

Proof Intuition. We now give a high-level (somewhat inaccurate) description of how the proof of security works. The simulator essentially runs rounds I and II honestly on behalf of the honest parties, but encrypts 0s instead of their real inputs. Then, in round III, it tries to force the real-world protocol output to match the idea-world output μ^*, by giving an incorrect decryption share on behalf of some honest party P_h. That is, assume that the combined ciphertext at the end of round II is $c = (\mathbf{v}, w, D)$. The simulator can get the secret keys $\mathbf{s}_D^k$ of all semi-malicious parties P_k at the end of round I (recall that semi-malicious parties follow the protocol honestly up to choosing bad random coins which are available to the simulator). It can therefore approximately compute the decryption shares $w^k \approx \langle \mathbf{v}, \mathbf{s}_D^k \rangle$ of the semi-malicious parties before (round III) starts. It then chooses the decryption share w^h of the honest party P_h by solving the equation $w - \sum_\ell \langle \mathbf{v}, w^\ell \rangle = 2e + \mu^*$ where $e \xleftarrow{\$} B_{smdg}^{dec}$ is added noise. The decrypted value is therefore μ^*. We claim that the simulation is "good" since:

- The way that the simulator computes the decryption share of party P_h is actually statistically close to the way that the decryption share is given in the real world, when the noise B_{smdg}^{dec} is large enough. This follows by the "smudging" lemma.
- The attacker cannot distinguish encryptions of 0 from the real inputs by the "security of joint keys" (Lemma 3.4). In particular, the combined public encryption-key pk is derived as the sum of an honestly generated public-key $\mathbf{p}_0^h$ (for party P_h) and several other honestly and semi-maliciously generated keys for which the attacker must "know" a corresponding secret key. Moreover, the secret key $\mathbf{s}_0^h$ of party P_h is now never used during the decryption protocol. Therefore, by the "security of joint keys", encryptions under pk maintain semantic security. There is an added complication here that extra information about the secret key $\mathbf{s}_0^h$ is released during rounds I and II of the protocol to create the evaluation key. However, this extra information essentially consists of ciphertexts under the higher level secret keys $\mathbf{s}_d^h$ for $d = 1, \ldots, D$. Therefore, the full proof consists of several hybrid games where we replace this extra information with random values starting with the top level and going down.

6 Variants and Optimizations

We consider several variants and optimizations of our basic MPC protocol from the previous section.

Randomized Functionalities and Individual Outputs. Our basic MPC protocol only considers deterministic functionalities where all the parties receive the same output. However, we can use standard efficient and round-preserving transformations to get a protocol for probabilistic functionalities and where different parties can receive different outputs. See Appendix D for details.

Security for Fully Malicious Attackers. Our basic MPC protocol is only secure in the semi-malicious setting. In Appendix E, we give a general round-preserving compiler from semi-malicious to fully malicious security using UC NIZKs [SCO+01] in the CRS model. In particular, in each round, the attacker must prove (in zero-knowledge) that it is following the protocol consistently with *some* setting of the random coins.

In Appendix F, we then turn to the question of *efficiency*. We first present simple, efficient, and statistical Σ-protocols for basic LWE-languages, along the lines of Schnorr's protocol [Sch91] (and bearing much similarity to Σ-protocols of [MV03] for various lattice problems). These Σ-protocols crucially rely on the idea of "smudging" and have an interesting caveat that there is a *gap* between the noise-levels for which zero-knowledge is shown to hold and the ones for which soundness holds. We then use the Σ-protocols for these basic LWE-languages along with a series of AND and OR proofs to convert them into Σ-protocols for the more complicated language showing that a party is following our MPC protocol specification honestly with some (not necessarily honestly chosen) random coins. Using the Fiat-Shamir heuristic [FS86] we then get efficient UC NIZKs, and therefore also general 3-round MPC protocols, in the random oracle model.

Two Rounds with PKI . An alternative way to present our protocol is as a 2-round protocol with a *public-key infrastructure* (PKI). In particular, we can think of the (round I) message $(\{\mathbf{p}_d^k\} , \{b_{d,i,\tau}^{\ell,k}\})$ sent by party P_k as its *public key* and the value $\{\mathbf{s}_D^k\}$ as its secret key (in the fully malicious setting, the public key would also contain the corresponding NIZKs). The entire MPC execution then only consists of the remaining two rounds. Note that this PKI is very simple and does not need a trusted party to set up everything; we just need a trusted party to choose a CRS and then each party can choose its own public key individually (possibly maliciously). Moreover, the PKI can be *reused* for many MPC executions of arbitrary functions f with arbitrary inputs. The main drawback is that the size of each party's public key is proportional to the total number of parties, and it would be interesting to remove this. The security analysis is exactly the same as that of our original three round protocol in the CRS model, just by noting that the first round there consists of broadcast message, which does not depend on the inputs of the parties (and hence we can think of it as a public key).

Bootstrapping. In our basic MPC protocol, the communication complexity is proportional to the maximal number of multiplicative-levels in the circuit of the evaluated function. This is because we start with a leveled TFHE scheme. To make the communication complexity completely independent of circuit size, we can rely on the *bootstrapping* technique of [Gen09]. In particular, since the TFHE scheme can evaluate any polynomial $D = D(\kappa)$ number of levels, it can also evaluate its own decryption circuit and some extra operations when we set D large enough.[9] To apply the

[9]This is not generically true for leveled FHE since the depth of decryption may depend on (and exceed) D so that the scheme would be unable to evaluate its own decryption circuit. However, in our case, *only* the dimension n depends *polynomially* on D, while the multiplicative depth of the decryption circuit only depends *poly-logarithmically* on n. Hence it is possible to choose a sufficiently large D so that the scheme can evaluate its own decryption.

bootstrapping technique, each party P_k only needs to encrypt its secret-key share $sk_k = \mathbf{s}_D^k$ (bit-by-bit) under the combined public-key pk in (round II) of the protocol, and we add these values to the *evaluation key*. With this modification, we can instantiate our TFHE scheme with some *fixed* polynomial D and maintain the ability to homomorphically evaluate arbitrarily large circuits with the same correctness guarantees as previously. Therefore, the communication/computation complexity of the key-generation and decryption protocols is completely independent of the circuit size of the function f. For security, however, we must now rely on a non-standard *circular-security assumption* for the basic LWE-based encryption scheme E.

Outsourcing Computation. The only intensive computation in our protocol, that depends on the circuit size of the evaluated function, is the homomorphic evaluation. In our basic description of the protocol, we assumed that each party performs this computation *individually* at the end of round II. However, it is also possible to designate one special party P^* (or even an external entity e.g. a powerful server, or the "cloud") that does this computation alone on everyone's behalf and broadcasts the resulting output ciphertexts to everyone else.[10] Moreover, P^* does not need to broadcast its input ciphertexts to everyone else, as it alone needs to know them when performing the evaluation. That is, the communication complexity is only proportional to the output size and the inputs of all parties *other* than P^*. This may be useful if the MPC computation involves one powerful party with a huge input and many weaker parties with small inputs.

The above simple idea already achieves security in the semi-honest model, where we can trust P^* to perform the computation honestly and return the correct result. However, in the fully malicious setting, we would also require P^* to prove that the resulting ciphertext is the correct one, using a computationally-sound proof system with a fixed polynomial (in the security parameter) verification complexity. Such non-interactive proofs are known to exist in the random-oracle model [Mic00] or under appropriate non-black-box assumptions [BCCT12, GLR11, DFH11].

Fairness. Our basic MPC protocol achieves security with abort for any number of corrupted parties. We can also achieve *fairness* for $t < N/2$ corruptions. The main idea is that, in Round I, each party also (threshold) secret-shares its individual secret sk_d^k so that any $\lfloor N/2 \rfloor + 1$ parties can recover it, but any fewer will not get any extra information. If a party P_k aborts in Rounds II or III, the rest of the parties will reconstruct sk_d^k (at the cost of one extra round) and use it to continue the protocol execution on P_k's behalf. Although an honest execution of our fair MPC protocol still uses 3 rounds of interaction, the protocol may now take up to 5 rounds in the worst case when some parties abort, where the extra rounds are needed to reconstruct the keys of the aborted parties. See Appendix G for details.

References

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, pages 595–618, 2009.

[10] Broadcasting the output ciphertexts requires an extra round, raising the round complexity to 4 rounds in the CRS model, 3 rounds in PKI model.

[AIK05] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. In *IEEE Conference on Computational Complexity*, pages 260–274, 2005.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, 2012.

[BD10] Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In *TCC*, pages 201–218, 2010.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, pages 169–188, 2011.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.

[BNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM Conference on Computer and Communications Security*, pages 257–266, 2008.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *FOCS*, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.

[CEMY09] Seung Geol Choi, Ariel Elbaz, Tal Malkin, and Moti Yung. Secure multi-party computation minimizing online rounds. In *ASIACRYPT*, pages 268–286, 2009.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369, 1986.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *CRYPTO*, pages 487–504, 2011.

[DFH11] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. Cryptology ePrint Archive, Report 2011/508, 2011. http://eprint.iacr.org/.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *CRYPTO*, pages 378–394, 2005.

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *CRYPTO*, pages 501–520, 2006.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *CRYPTO*, pages 247–264, 2003.

[FH96] Matthew K. Franklin and Stuart Haber. Joint encryption and message-efficient secure computation. *J. Cryptology*, 9(4):217–232, 1996.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GH11] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *EUROCRYPT*, pages 129–148, 2011.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *CRYPTO*, pages 155–172, 2010.

[GKPV10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In *ICS*, pages 230–240, 2010.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. Cryptology ePrint Archive, Report 2011/456, 2011. http://eprint.iacr.org/.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 1st edition, 2004.

[Goy11] Vipul Goyal. Constant round non-malleable protocols using one way functions. In *STOC*, pages 695–704, 2011.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO*, pages 132–150, 2011.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

[JJ00] Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts. In *ASIACRYPT*, pages 162–177, 2000.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *CRYPTO*, pages 335–354, 2004.

[KOS03] Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Round efficiency of multi-party computation with a dishonest majority. In *EUROCRYPT*, pages 578–595, 2003.

[Lin01] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In *CRYPTO*, pages 171–189, 2001.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[LP11] Huijia Lin and Rafael Pass. Constant-round non-malleable commitments from any one-way function. In *STOC*, pages 705–714, 2011.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

[MSS11] Steven Myers, Mona Sergi, and Abhi Shelat. Threshold fully homomorphic encryption and secure computation. In *eprint 2011/454*, 2011.

[MV03] Daniele Micciancio and Salil P. Vadhan. Statistical zero-knowledge proofs with efficient provers: Lattice problems and more. In *CRYPTO*, pages 282–298, 2003.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *STOC*, pages 333–342, 2009.

[RAD78] R.L. Rivest, L. Adleman, and M.L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations on Secure Computation, Academia Press*, pages 169–179, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[SCO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, pages 566–598, 2001.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography*, pages 420–443, 2010.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

A Definitions

A.1 The Universal Composability Framework (UC)

We work in the standard universal composability framework of Canetti [Can01] with static corruption. The UC framework defines a PPT environment $\mathcal{Z}$ that is invoked on security parameter 1^κ and an auxiliary input z, and oversees the execution of a protocol in one of two worlds. The "ideal world" executions involves dummy parties $\tilde{P}_1, \ldots, \tilde{P}_N$, 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. The environment $\mathcal{Z}$ chooses the inputs of the parties, may interact with the ideal / real adversary during the execution, and at the end of the execution need to decide whether a real or ideal execution has been taken place. The output of the execution is simply the output of the environment. We refer to [Can01] for further details.

Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa, z)$ denote the random variable describing the output of the environment $\mathcal{Z}$ after interacting in the ideal process with adversary $\mathcal{S}$, the functionality $\mathcal{F}$, on security parameter 1^κ and input z. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$. Similarly, let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(1^\kappa, z)$ denote the random variable describing the output of the environment $\mathcal{Z}$ after interacting with the adversary $\mathcal{A}$ and parties running protocol π on security parameter κ, and input z. Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(1^\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$.

Definition A.1. *For $N \in \mathbb{N}$, let $\mathcal{F}$ be an N-ary functionality, and let π be a N-party protocol. We say that π* securely realizes *$\mathcal{F}$ if for any* PPT *adversary $\mathcal{A}$ there exists a* PPT *ideal adversary $\mathcal{S}$ such that for any* PPT *environment $\mathcal{Z}$ we have:*

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \stackrel{c}{\equiv} \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$$

We sometime want to restrict the definition so that it quantifies over adversaries from a certain class: semi-honest adversaries, or malicious adversary that corrupted only a certain number of parties. That is done in a straightforward way.

General functionality. We consider the general UC–functionality $\mathcal{F}$, which securely evaluates any function $f : (\{0,1\}^{\ell_{in}})^N \to (\{0,1\}^{\ell_{out}})^N$. The functionality $\mathcal{F}_f$ is parameterized with a function f and is defined as follows:

- Each party P_i sends (x_i, sid) to the functionality.
- Once all parties send their inputs, evaluate $(y_1, \ldots, y_N) \leftarrow f(x_1, \ldots, x_N)$.
- Send to each party P_i the output (y_i, sid).

Fairness. Our default notion is "security-with-abort" meaning that the ideal adversary (simulator) can abort the computation and cause the functionality to not give output. In addition, we say that a protocol π securely and fairly realizes a functionality $\mathcal{F}$ if $\mathcal{S}$ does not get the ability to abort. Meaning, the functionality always sends to the honest party the outputs.

A.2 Security Against Semi-Malicious Adversaries

As a stepping stone towards realizing the standard definition of secure multi-party computation against active adversaries, we define a notion of a *semi-malicious* adversary that is stronger than the standard notion of semi-honest adversary. We formalize security against semi-malicious adversaries, and later in Section E, we present a generic transformation from an MPC protocol that is secure against semi-malicious adversaries, to a protocol that is secure against fully malicious adversaries.

Semi–malicious adversary. A *semi-malicious adversary* is modeled as an interactive Turing machine (ITM) which, in addition to the standard tapes, has a special *witness tape*. In each round of the protocol, whenever the adversary produces a new protocol message m on behalf of some party P_k, it must also write to its special witness tape *some* pair (x, r) of input x and randomness r that *explains its behavior*. More specifically, all of the protocol messages sent by the adversary on behalf of P_k up to that point, including the new message m, must exactly match the honest protocol specification for P_k when executed with input x and randomness r. Note that the witnesses given in different rounds need not be consistent. Also, we assume that the attacker is rushing and hence may choose the message m and the witness (x, r) in each round adaptively, after seing the protocol messages of the honest parties in that round (and all prior rounds). Lastly, the adversary may also choose to abort the execution on behalf of P_k in any step of the interaction.

Definition A.2. *We say that a protocol π* securely realizes *$\mathcal{F}$* for semi-malicious adversaries *if it satisfies Definition A.1 when we only quantify over all semi-malicious adversaries $\mathcal{A}$.*

We remark that this definition captures the semi-honest adversary who always follows the protocol with honestly chosen random coins (and can be easily modified to write those on its witness tape). On the other hand, a semi-malicious adversary is more restrictive than a fully malicious adversary, since its behavior follows the protocol with *some* input and randomness which it must *know*. Note that the semi-malicious adversary may choose the different input or random tape in an adaptive fashion using any PPT strategy according to the partial view it has seen.

Discussion. The main benefit in considering semi-malicious adversaries is shown in Appendix E. In particular, we get a simple general compiler from semi-malicious security to fully malicious security, just by having each party in each round give a ZK proof that it is following the protocol honestly with some input x and randomness r. Unlike the general compiler from semi-honest security to fully malicious security, we *do not* need to coin-flip for the random coins of the parties.

B FHE Scheme of [BV11a, BGV12]

Modulus Reduction. Before describing the scheme, let us start with a basic lemma from [BGV12] called the "modulus switching lemma".

Lemma B.1 (Modulus Switching [BGV12])**.** *Let $q \geq p$ be two odd moduli and let $\mathbf{c}$ be an integer vector. Define $\mathbf{c}'$ to be the integer vector closes to $(p/q)\mathbf{c}$ such that $\mathbf{c}' = \mathbf{c} \pmod 2$. Then, for any $\mathbf{s}$ with $|[\langle \mathbf{c}, \mathbf{s}\rangle]_q| < q/2 - (q/p) \cdot \ell_1(\mathbf{s})$, we have:*

$$[\langle \mathbf{c}', \mathbf{s}\rangle]_p = [\langle \mathbf{c}, \mathbf{s}\rangle]_q \pmod 2 \qquad \textit{and} \qquad |[\langle \mathbf{c}', \mathbf{s}\rangle]_p| < (p/q) \cdot |[\langle \mathbf{c}, \mathbf{s}\rangle]_q| + \ell_1(\mathbf{s})$$

The lemma shows that, if we use the basic LWE-based encryption scheme E with some modulus q and a secret key $\mathbf{s}$ which is "sufficiently small", we can take a ciphertext $\mathbf{c} = (\mathbf{v}, w)$ and "scale it down" by a factor of p/q, to get a new valid ciphertext under the same key $\mathbf{s}$ (encrypting the same message) but using a smaller modulus p. Moreover, the absolute value of the noise decreases by essentially a factor of p/q. This lemma is used to keep the noise level "small", by simply scaling down the ciphertext to a smaller modulus before each multiplication operation.

The Construction We now define the homomorphic encryption scheme of [BGV12, BGV12] in terms of the basic LWE-based encryption scheme E from Section 3. We start by only describing the "leveled homomorphic encryption" which can evaluate any circuit with up to D multiplicative levels, where D is arbitrary but known to the key-generation algorithm. To get a "fully" homomorphic encryption where a single key-generation algorithm is sufficient to evaluate any polynomial number of levels, we can then use the "bootstrapping technique" of [Gen09] (requiring an additional circular security assumption).

The scheme defined as follows:

Parameters: $\mathsf{params} = \{\mathsf{params}_d : 0 \leq d \leq D\}$, where params_d is the tuple $(1^\kappa, q_d, m, n, \varphi, \chi)$ as in the basic scheme E. Note that only the modulus q_d differs for each "level" d.

FHE.Keygen(params) : The key generation algorithm creates (pk, sk, evk) as follows.

- **The evaluation key:** For every level $d \in \{0, \ldots, D\}$, choose a key $\mathbf{s}_d \leftarrow \mathsf{E.SymKeygen}(\mathsf{params}_d)$. Then, for all $d \in \{1, \ldots, D\}$, $i \in [n], j \in [n] \cup \{0\}$ and $\tau \in \{0, \ldots, \lfloor \log q_d \rfloor\}$ compute:

  $$\psi_{d,i,j,\tau} \stackrel{\text{def}}{=} (\mathbf{a}_{d,i,j,\tau}, b_{d,i,j,\tau}) \leftarrow \mathsf{E.SymEnc}_{\mathbf{s}_d}(2^\tau \cdot \mathbf{s}_{d-1}[i] \cdot \mathbf{s}_{d-1}[j])$$

  so that

  $$b_{d,i,j,\tau} = \langle \mathbf{a}_{d,i,j,\tau}, \mathbf{s}_d\rangle + 2 \cdot e_{d,i,j,\tau} + 2^\tau \cdot \mathbf{s}_{d-1}[i] \cdot \mathbf{s}_{d-1}[j]$$

  for some coefficients $\mathbf{a}_{d,i,j,\tau}$ and noise $e_{d,i,j,\tau}$. The above is an approximate encryption of $\mathbf{s}_{d-1}[i] \cdot \mathbf{s}_{d-1}[j]$ under $\mathbf{s}_d$, where $\mathbf{s}_{d-1}[0] \stackrel{\text{def}}{=} 1$. We define $evk = \{(\mathbf{a}_{d,i,j,\tau}, b_{d,i,\tau})\}_{d,i,j,\tau}$.

- **The public key:** Invoke $(A, \mathbf{p}) \leftarrow \mathsf{E.PubKeygen}(\mathbf{s}_0)$ and set $pk = (A, \mathbf{p})$. Note that $\mathbf{p} = A\mathbf{s}_0 + 2\mathbf{e}$ for some noise vector $\mathbf{e}$.

- **The secret key:** The secret key is simply $sk \stackrel{\text{def}}{=} \mathbf{s}_D$.

$\mathsf{FHE.Enc}_{pk}(\mu)$: The encryption algorithm computes $(\mathbf{v}, w) \leftarrow \mathsf{E.PubEnc}_{pk}(\mu)$ and outputs $(\mathbf{v}, w, 0)$ to indicate that the "level" of the ciphertext is 0. Note that:

$$\mathbf{v} = \mathbf{r}^T \cdot A, \qquad w = \langle \mathbf{r}, \mathbf{p} \rangle + \mu \quad (\ = \langle \mathbf{v}, \mathbf{s}_0 \rangle + 2\langle \mathbf{r}, \mathbf{e} \rangle + \mu \text{ when } \mathbf{p} = A\mathbf{s}_0 + 2\mathbf{e} \quad)$$

for some random vector $\mathbf{r} \in \{0,1\}^m$.

$\mathsf{FHE.Dec}_{sk}(c)$: On input $c = (\mathbf{v}, w, D)$ the decryption algorithm returns:

$$\mathsf{E.Dec}_{sk}((\mathbf{v}, w)) = [w - \langle \mathbf{v}, \mathbf{s}_D \rangle]_{q_D} \pmod 2.$$

$\mathsf{FHE.Eval}_{evk}(f, c_1, \ldots, c_t)$: Let f be a function computed by a boolean circuit C which consists of '+' gates and '$\times$' gates with fan-in 2. Furthermore, assume the circuit is *layered* meaning that it is composed of levels consisting either entirely of '+' gates or entirely of '$\times$' gates, and that the number of multiplicative '$\times$' layers is bounded by D. We associate the input ciphertexts with the input wires and homomorphically evaluate the circuit gate by gate. Given two ciphertexts c_1, c_2, where $c_1 = ((\mathbf{v}_1, w_1), d)$ and $c_2 = ((\mathbf{v}_2, w_2), d)$, having matching "levels" d, we can perform the following operations:

- *Addition gates.* The resulting ciphertext is:

$$c_{\mathsf{add}} = ((\mathbf{v}_{\mathsf{add}}, w_{\mathsf{add}}), d) \stackrel{\text{def}}{=} ((\mathbf{v}_1 + \mathbf{v}_2, w_1 + w_2), d)$$

- *Multiplication gates.* As a first step, we do a "*modulus switch*" on the values $(\mathbf{v}_1, w_1), (\mathbf{v}_2, w_2)$ to convert them from ciphertexts over the modulus q_d to ones over the (smaller) modulus q_{d+1}. To do so define the "scaling factor" $\beta_d := (q_d/q_{d+1}) \in \mathbb{R}$. Then, for $b \in \{1,2\}$, let $(\mathbf{v}'_b, w'_b)$ be the integer vector which is closest to $(1/\beta_d) \cdot (\mathbf{v}_b, w_b)$ in ℓ_1-norm and which satisfies $(\mathbf{v}'_b, w'_b) = (\mathbf{v}_b, w_b) \pmod 2$ component-wise.

  Next, consider the n-variate symbolic polynomial over the unknown vector $\mathbf{x}$ given by:

$$\begin{aligned} \phi_{mult}(\mathbf{x}) &\stackrel{\text{def}}{=} (w'_1 - \langle \mathbf{v}'_1, \mathbf{x} \rangle) \cdot (w'_2 - \langle \mathbf{v}'_2, \mathbf{x} \rangle) \\ &= \sum_{0 \le i \le j \le n} h_{i,j} \cdot \mathbf{x}[i] \cdot \mathbf{x}[j] \end{aligned}$$

  for some coefficients $h_{i,j}$ over $\mathbb{Z}_{q_{d+1}}$.[11] We can expand it more as:

$$\phi_{mult}(\mathbf{x}) = \sum_{\substack{0 \le i \le j \le n \\ \tau \in \{0, \ldots, \lfloor \log q_{d+1} \rfloor\}}} h_{i,j,\tau} \cdot 2^\tau \cdot \mathbf{x}[i] \cdot \mathbf{x}[j]$$

  where $(h_{i,j,1}, \ldots, h_{i,j,\lfloor \log q_{d+1} \rfloor})$ is the binary representation of $h_{i,j}$. Then, we set:

$$c' \stackrel{\text{def}}{=} \sum_{\substack{0 \le i \le j \le n \\ \tau \in \{0, \ldots, \lfloor \log q_{d+1} \rfloor\}}} h_{i,j,\tau} \cdot \psi_{d+1,i,j,\tau}$$

[11] Recall that we define $\mathbf{x}[0] = 1$.

and set $c_{\mathsf{mult}} \stackrel{\text{def}}{=} (c', d+1)$. [12] Notice that if $\phi_{c'}$ is the symbolic polynomial associated with c' then

$$\begin{aligned}\phi_{c'}(\mathbf{s}_{d+1}) &= \sum_{\substack{0 \le i \le j \le n \\ \tau \in \{0,\ldots,\lfloor \log q_{d+1} \rfloor\}}} h_{i,j,\tau} \cdot \phi_{\psi_{d,i,j,\tau}}(\mathbf{s}_{d+1}) \\ &\approx \sum_{\substack{0 \le i \le j \le n \\ \tau \in \{0,\ldots,\lfloor \log q_{d+1} \rfloor\}}} h_{i,j,\tau} 2^{\tau} \mathbf{s}_d[i]\mathbf{s}_d[j] = \phi_{mult}(\mathbf{s}_d) = \phi_{c_1}(\mathbf{s}_d) \cdot \phi_{c_2}(\mathbf{s}_d)\end{aligned}$$

where the "$\approx$" hides the sum of the noises in the ciphertexts $\psi_{d,i,j,\tau}$. Therefore the ciphertext c' encrypts the product of the plaintexts associated with c_1, c_2, but under the new key $\mathbf{s}_{d+1}$ and the new (smaller) modulus q_{d+1}.

Upgrades. If we want to perform operations on two ciphertexts $c_1 = ((\mathbf{v}_1, w_1), d_1)$ and $c_2 = ((\mathbf{v}_2, w_2), d_2)$, having *differing* levels $d_2 > d_1$, we first "upgrade" the level of c_1 to d_2 and then perform operations as above. To do the upgrade, we can just evaluate sufficiently mane extra "dummy" multiplication gates where we multiply the plaintext by 1. We use the fixed ciphertexts $(\mathbf{v} = (0,\ldots,0), w = 1, d)$ as "dummy" encryptions of 1.

Correctness. We now show that the scheme satisfies correctness for the following parameters.

Proposition B.2. *Assume that* $\mathsf{params} = \{\mathsf{params}_d = (1^\kappa, q_d, m, n, \varphi, \chi) : d \in \{0,\ldots,D\}\}$ *are chosen so that the moduli* $q_0,\ldots,q_D$ *satisfy* $q_{d-1}/q_d \ge \rho$ *for all* $d \in [D]$*, the noise* φ *is* B_φ*-bounded, and* χ *is* B_χ*-bounded. Further assume that:*

$$q_D \ge 2\rho \cdot (nB_\varphi + 2) \quad , \quad \rho \ge 2^{\omega(\log(\kappa))} \cdot \max\{B_\varphi^2, B_\chi\}.$$

Then FHE *satisfies the correctness property of leveled fully homomorphic encryption.*

Actually, we will need to prove something more general. Fix $\mathsf{params} = \{\mathsf{params}_d : d \in \{0,\ldots,D\}\}$, *any* evaluation key $evk = \{(\mathbf{a}_{d,i,j,\tau}, b_{d,i,j,\tau})\}_{d,i,j,\tau}$, any ciphertexts $\{c_i = (\mathbf{v}_i, w_i, 0)\}$, and message-bits $\{\mu_i\}$ for which that there exist *some* values $\{\mathbf{s}_d\}$, $\{e_{d,i,j,\tau}\}$, $\{e_i\}$ satisfying:

$$\begin{aligned} b_{d,i,j,\tau} &= \langle \mathbf{a}_{d,i,j,\tau}, \mathbf{s}_d \rangle + 2 \cdot e_{d,i,j,\tau} + 2^\tau \cdot \mathbf{s}_{d-1}[i] \cdot \mathbf{s}_{d-1}[j] \\ w_i &= \langle \mathbf{v}_i, \mathbf{s}_0 \rangle + 2e_i + \mu_i \end{aligned}$$

with bounds $\ell_1(\mathbf{s}_d) \le B_{sec}$, $|e_i| \le B_{enc}$, $|e_{d,i,j,\tau}| \le B_{eval}$ for $B_{sec}, B_{enc}, B_{eval} \in \mathbb{Z}$.

Proposition B.3. *Let* $(\mathsf{params}, evk, sk, \{c_i\}, \{\mu_i\})$ *satisfy the above requirements and let* ρ *be some value such that* $q_{d-1}/q_d \ge \rho$ *for all* $d \in [D]$*, and*

$$q_D \ge 2\rho \cdot (B_{sec} + 2) \quad , \quad \rho \ge 2^{\omega(\log(\kappa))} \cdot \max\{B_{sec}^2, B_{enc}, B_{eval}\}.$$

Let f *be some boolean function whose circuit has at most* D *multiplicative levels, and let* $c^* = \mathsf{FHE.Eval}_{evk}(f, c_1, \ldots, c_t)$*. Then* $\mathsf{FHE.Dec}_{sk}(c^*) = f(\mu_1, \ldots, \mu_t)$*. Furthermore,* $\mathsf{noise}(c^*, \mathbf{s}_D) \le \rho$*.*

[12] Above, we implicitly define $\psi_{d+1,0,0,\tau} := (0^n, 2^\tau)$ and $\psi_{d+1,i=0,j,\tau} := \psi_{d,j,0,\tau}$. Notice that this retains consistency of $\psi_{d+1,i,j,\tau}$ "aproximately encrypting" $2^\tau \mathbf{s}_d[i][j]$ under the key $\mathbf{s}_{d+1}$.

Proof: We look at the *noise* of the intermediate ciphertexts created during evaluation of the circuit and inductively show that its magnitude never exceeds ρ and its parity corresponds to the correct bit for the corresponding wire in the circuit.

Initial Noise. For the initial ciphertexts c_i we have $\mathsf{noise}_{q_0}(c_i, \mathbf{s}_0) \leq B_{init}$ where $B_{init} := (2B_{enc} + 1) \leq \rho$ and the parity of the noise is μ_i by assumption.

Multiplicative Noise. Let $c_1 = ((\mathbf{v}_1, w_1), d)$ and $c_2 = ((\mathbf{v}_2, w_2), d)$ be two ciphertexts such that $\mathsf{noise}_{q_d}(c_b, \mathbf{s}_d) \leq \rho$ and where the parity of the noise is μ_1, μ_2 respectively. Let c_{mult} be the resulting ciphertext after evaluating a multiplication gate on c_1, c_2.

- Firstly, let us consider the ciphertexts c_1', c_2' produced by performing "modulus reduction". Namely, by applying Lemma B.1, we see that

  $$\mathsf{noise}_{q_{d+1}}(c_1', \mathbf{s}_d) \leq \mathsf{noise}_{q_d}(c_1, \mathbf{s}_d)/\rho + \ell_1(\mathbf{s}_d) + 1 \leq B_{sec} + 2$$

  and similarly for c_2'. Further, the parity of the noise remains the same.

- Now consider the symbolic polynomial $\phi(\mathbf{x})$ over $\mathbb{Z}_{q_{d+1}}[\mathbf{x}]$. If we define $y = \phi(\mathbf{s}_d)$ then $|[y]_{q_{d+1}}| \leq (B_{sec} + 2)^2$ and $[y]_{q_{d+1}} = \mu_1 \cdot \mu_2 \mod 2$.

- Lastly, let

  $$c_{mult} \stackrel{\text{def}}{=} (\mathbf{v}_{mult}, w_{mult}) = \sum_{\substack{0 \leq i \leq j \leq n \\ \tau \in \{0, \ldots, \lfloor \log q_{d+1} \rfloor\}}} h_{i,j,\tau} \cdot (\mathbf{a}_{d,i,j,\tau}, b_{d,i,j,\tau})$$

  where the coefficients $h_{i,j,\tau}$ are given by the expansion of ϕ. Then:

  $$\begin{aligned} y' &\stackrel{\text{def}}{=} w_{mult} - \langle \mathbf{v}_{mult}, \mathbf{s}_{d+1} \rangle = \sum_{i,j,\tau} h_{i,j,\tau} \cdot (b_{d,i,j,\tau} - \langle \mathbf{a}_{d,i,j,\tau}, \mathbf{s}_{d+1} \rangle) \\ &= \sum_{i,j,\tau} h_{i,j,\tau} \cdot (\ 2^\tau \cdot \mathbf{s}_{d-1}[i] \cdot \mathbf{s}_{d-1}[j] \ + \ 2 \cdot e_{d,i,j,\tau}\ ) = y + 2 \sum_{i,j,\tau} h_{i,j,\tau} \cdot e_{d,i,j,\tau} \end{aligned}$$

  Therefore $\mathsf{noise}_{q_{d+1}}(c_{mult}, \mathbf{s}_{d+1}) \leq B_{mult}$ where

  $$B_{mult} := (B_{sec} + 2)^2 + 2(n+1)^2(\lfloor \log q_{d+1} \rfloor + 1) \cdot B_{eval} \leq \rho$$

  and its parity again remains $\mu_1 \cdot \mu_2$.

Additive Noise. We can assume that there are at most $\gamma = \mathsf{poly}(\kappa)$ successive levels of addition. Then the noise of any ciphertext c_{add} which corresponds to the output of an addition gate is bounded by $B_{add} := \gamma \cdot \max\{B_{mult}, B_{init}\}$. Since $B_{init}, B_{mult} = \rho/2^{\omega(\log \kappa)}$ we see that $B_{add} \leq \rho$ as we wanted to show. ∎

Proof of Proposition B.2. Notice that, if the distributions φ, χ are B_φ, B_χ bounded respectively and the values $(pk, evk, sk) \leftarrow \mathsf{FHE.Keygen}(\mathsf{params})$, $\{c_i \leftarrow \mathsf{FHE.Enc}_{pk}(\mu_i)\}$ are chosen honestly then the above requirements on $(\mathsf{params}, evk, sk, c_1, \ldots, c_t, \mu_1, \ldots, \mu_t)$ are satisfied with $B_{sec} = n \cdot B_\varphi$, $B_{eval} = B_\chi$ and $B_{enc} = m \cdot B_\chi$. ∎

C Proof of Security for TFHE-Based MPC

C.1 Correctness of TFHE Protocol

Let us assume that:

1. The scheme TFHE is honestly initialized with some setup.
2. The N-party protocol $\mathsf{TFHE.Keygen}$ is executed according to the protocol specification but with *some arbitrary* (worst-case) random coins, resulting in the common public outputs pk, evk and a secret-key share sk_k for party P_k. Let $sk^* := \sum_{k=1}^{N} sk_k$.
3. The ciphertexts $\{c_i\}$ are chosen as encryptions of the bits $\{\mu_i\}$ via $\mathbf{c}_i \leftarrow \mathsf{TFHE.enc}_{pk}(\mu_i)$, but again using *some arbitrary* (worst-case) random coins for encryption.
4. Let f be some boolean function whose circuit has at most D levels and let $c^* = \mathsf{TFHE.Eval}_{evk}(f, c_1, \ldots, c_t)$.
5. The N-party protocol $\mathsf{TFHE.Dec}$ is executed on the ciphertext c^* as common input and the individual keys sk_k for party P_k, where each party follows the protocol specification with *some arbitrary* (worst-case) random coins. Let μ^* be the common output.

Recall that, in all above steps, we assume that any sample from $x \leftarrow \chi$ (resp. $x \leftarrow \varphi$) is bounded by $|x| \le B_\chi$ (resp. $|x| \le B_\varphi$). We make one additional *generalization* and assume that whenever the protocol specifies a value $r \xleftarrow{\$} \{0,1\}^m$ is to be chosen in the above steps (i.e. during second round of key generation and for encryption), we are only guaranteed $r \in [-B_r, B_r]^m$ for some positive integer bound B_r.

Theorem C.1. *Let* params *be chosen so that, for some* ρ*, we have* $q_{d-1}/q_d \ge \rho$ *for all* $d \in [D]$ *and:*

$$\begin{aligned} \rho &\ge 2^{\omega(\log(\kappa))} \cdot \max\left\{ B^{enc}_{smdg} \ , \ N^2 B^{eval}_{smdg} \ , \ N^2 n^2 B^2_\varphi \ , \ N^3 B_\chi B_\varphi \ , \ N^3 m B_r B_\chi \right\} \\ q_D &\ge \max\left\{ \ 2\rho \cdot (NnB_\varphi + 2) \ , \ 2(\rho + 2NB^{dec}_{smdg} + 1) \ \right\} \end{aligned}$$

Then

$$\mu^* = \mathsf{E.Dec}_{sk^*}(c^*) = f(\mu_1, \ldots, \mu_t) \quad \textit{and} \quad \mathsf{noise}_{q_D}(c^*, sk^*) \le \rho.$$

Proof: We rely on Proposition B.3 which shows the correctness of the underlying FHE scheme. In particular, when the key-generation protocol is executed as specified (with worst-case random coins), we get the combined secret keys $\mathbf{s}^*_d = \sum_{\ell=1}^{N} \mathbf{s}^\ell_d$ for each level d satisfying

$$\ell_1(\mathbf{s}^*_d) \le B_{sec} \stackrel{\text{def}}{=} NnB_\varphi.$$

As for the errors in the evaluation key, we have:

- The combined values $(\mathbf{a}^\ell_{d,i,\tau}, \beta^\ell_{d,i,\tau})$ defined at the end of Round 1 "approximately encrypt" $2^\tau \mathbf{s}^\ell_{d-1}[i]$ under $\mathbf{s}^*_d$ with error $|e| \le N \cdot B_\chi$.
- The values $(\boldsymbol{\alpha}^{\ell,k}_{d,i,j,\tau}, \beta^{\ell,k}_{d,i,j,\tau})$ sent out in Round 2 are "approximate encryptions" of $2^\tau \mathbf{s}^\ell_{d-1}[i]\mathbf{s}^k_{d-1}[j]$ under $\mathbf{s}^*_d$ with error $|e| \le NB_\chi B_\varphi + NmB_rB_\chi + B^{eval}_{smdg}$.

- The combined values $\psi_{d,i,j,\tau}$ defined at the end of Round 2 are "approximate encryptions" of $2^\tau \mathbf{s}^*_{d-1}[i]\mathbf{s}^*_{d-1}[j]$ under $\mathbf{s}^*_d$ with error

$$|e| \le B_{eval} \stackrel{\text{def}}{=} N^2(NB_\chi B_\varphi + NmB_r B_\chi + B^{eval}_{smdg}).$$

Finally, the ciphertexts c_i are encryptions of μ_i under $\mathbf{s}^*_0$ with error

$$|e| \le B_{enc} \stackrel{\text{def}}{=} mNB_r B_\chi + B^{enc}_{smdg}.$$

By applying Proposition B.3 with $B_{sec}, B_{eval}, B_{enc}$ as defined above, we get $c^* = (\mathbf{v}, w)$ such that $\mathsf{E.Dec}_{sk^*}(c^*) = \mu^*$ and $\mathsf{noise}_{q_D}(c^*, sk^*) \le \rho$. If each party correctly follows the decryption protocol with some randomness and gives the decryption share $w^k \stackrel{\text{def}}{=} \langle \mathbf{v}, \mathbf{s}^k_D \rangle + 2 \cdot e_k$ for some noise $e_k \stackrel{\$}{\leftarrow} [-B^{dec}_{smdg}, B^{dec}_{smdg}]$ then

$$w - \sum_{i=1}^{N} w^k = \mu^* + 2e^* - 2\sum_{k=1}^{N} e_k$$

Therefore, the decryption protocol recovers μ^* as long as $\rho + NB^{dec}_{smdg} + 1 < q/2$ which follows from the assumption of the theorem. ■

C.2 Security of MPC in the Semi-Malicious Setting

We now prove the security of our MPC protocol in the semi-malicious setting (Theorem 5.2). Before we do so, let us describe the restrictions on the parameters params for which correctness/security hold and which also correspond to the parameters of the LWE assumption needed for security.

Parameters (General). For full generality we can show the correctness/security of our scheme for any choice of $\mathsf{params} = \left(\{\mathsf{params}_d = (1^\kappa, q_d, m, n, \varphi, \chi)\}_{0 \le d \le D}\ ,\ B_\varphi, B_\chi, B^{eval}_{smdg}, B^{enc}_{smdg}, B^{dec}_{smdg}\right)$ satisfying the following conditions:

1. Firstly, the parameters need to satisfy the conditions of Theorem C.1 for correctness so that the moduli q_d are sufficiently *large* in relation to the noise bounds. In particular we need some value ρ such that $\{\ q_{d-1}/q_d \ge \rho\ \}_{d \in [D]}$ and

$$\begin{aligned} \rho &\ge 2^{\omega(\log(\kappa))} \cdot \max\left\{B^{enc}_{smdg}\ ,\ N^2 B^{eval}_{smdg}\ ,\ N^2 n^2 B^2_\varphi\ ,\ N^3 B_\chi B_\varphi\ ,\ N^3 m B_\chi\right\} \\ q_D &\ge \max\left\{\ 2\rho \cdot (NnB_\varphi + 2)\ ,\ 2(\rho + 2NB^{dec}_{smdg} + 1)\ \right\} \end{aligned}$$

2. Secondly, we need the "smudging" noise to be sufficiently *large*. Specifically:

$$B^{dec}_{smdg} \ge 2^{\omega(\log(\kappa))} \cdot \rho \quad , \quad B^{enc}_{smdg}, B^{eval}_{smdg} \ge 2^{\omega(\log(\kappa))} \cdot B_\chi.$$

3. Thirdly, we need the parameters $\mathsf{params}_d = (n, m, q_d, \varphi, \chi)$ to be chosen so that the underlying encryption scheme E is semantically secure with pseudorandom ciphertexts when instantiated with params_d for all $d \in \{0, \ldots, D\}$. Furthermore, we need the public-key to uniquely determine a secret key as in Claim 2.3. In particular:
   - We set $m \ge (n+1)\log(q_0) + \omega(\log(\kappa))$.
   - All moduli q_d are odd *primes*.

Given params as above, security will follow if the $\mathrm{LWE}_{n, q_d, \varphi, \chi}$ assumption holds for all $d \in \{0, \ldots, D\}$.

Parameters (Concrete). For concreteness, we can choose parameters as follows:

- Set $B_\varphi, B_\chi := 2^\kappa$, $B^{eval}_{smdg}, B^{enc}_{smdg} := 2^{2\kappa}$, $B^{dec}_{smdg} = 2^{4\kappa}$.
- Choose primes $q_d \in [2^{5\kappa(D-d+1)}, 2^{5\kappa(D-d+1)+1}]$. This ensures: $q_D \geq 2^{5\kappa}$, $q_{d-1}/q_d \geq 2^{3\kappa}$.
- Choose a sufficiently large $n = n(\kappa)$ and any distributions χ, φ which are 2^κ-bounded so that the $\mathrm{LWE}_{n,q_d,\varphi,\chi}$ is hard for all $d \in \{0, \ldots, D\}$.

Note that the hardness of the LWE problem depends on the *ratio* of the modulus q to the noise B_χ, and on the dimension n. In our case, the ratio of modulus to noise is exponentially large, but we are free to choose n so that 2^n is a *bigger* exponential. For example, by choosing $n = \kappa^2$, the security of our scheme can be based on the worst-case (quantum) hardness of getting sub-exponential $2^{O(\kappa)} = 2^{O(\sqrt{n})}$ approximation to GapSVP on n dimensional lattices in sub-exponential $2^{O(\sqrt{n})}$ time [Reg05, Pei09].

Theorem C.2 (Theorem 5.2, restated)**.** *Let f be any deterministic poly-time function with N inputs and single output. Let* params *satisfy the above general constraints and assume that the corresponding* $\mathrm{LWE}_{n,q_d,\varphi,\chi}$ *assumptions holds. Then the protocol π_f securely UC-realizes the ideal functionality $\mathcal{F}_f$ in the presence of a static* semi-malicious *adversary (Definition A.2) corrupting any $t \leq N$ parties.*

Proof: Let $\mathcal{A}$ be a static semi-malicious adversary, and let I be the set of corrupted parties. Since simulating the case $I = [N]$ is trivial, we assume that $t = |I| \leq N - 1$. Let $h \notin I$ be some arbitrary index of an honest party P_h. We first assume a restricted semi-malicious adversary that is not allowed to abort. As a result, the simulator is not allowed to abort as well. After proving security with respect to this kind of adversary, we add a simple extension to the general case with aborts.

The Simulator. We start with an informal description of the simulator. For the execution of the $\mathsf{TFHE.Keygen}$ protocol, it simulates all the honest parties according to the honest protocol, except for P_h, for which it just sends uniformly random values. It then encrypts 0s instead of the real input bits of all honest parties. Finally, for the execution of the $\mathsf{TFHE.Dec}$ protocol on some ciphertext c^*, the simulator again simulates all of the honest parties honestly *except* for P_h, for which it sends some value so as to "fix" the decrypted plaintext to the correct output. We describe this formally.

- For every honest party $k \notin I$, $k \neq h$, simulate P_k by following the $\mathsf{TFHE.Keygen}$ and $\mathsf{TFHE.Dec}$ protocols honestly in all rounds.
- For every honest party $k \notin I$ (including P_h), choose the ciphertexts $c_{k,i}$ as $c_{k,i} \leftarrow \mathsf{TFHE.Enc}_{pk}(0)$ to be encryptions of 0 instead of encrypting the actual input bits $\mathbf{x}_k[i]$.
- For the special party P_h, proceed as follows:
  - During the execution of Keygen, the simulator $\mathcal{S}$ chooses the values $\{b^{h,\ell}_{d,i,j,\tau}\}_{d,i,j,\tau,\ell}$ and $\{\gamma^{\ell,h}_{d,i,j,\tau}\}_{d,i,j,\tau,\ell}$ on behalf of the party P_h *uniformly at random* instead of as specified, for every level $d \in [D]$. It still chooses the public-key shares $\{\mathbf{p}^h_d\}_{d\in\{0,\ldots,D\}}$ honestly.

– At the conclusion of the second round, the simulator reads the special "witness tape" of the semi-malicious adversary to learn the values $\mathbf{s}_d^i$ and the inputs $\mathbf{x}_i$ for $i \in I$. It sends the values $\{\mathbf{x}_i\}_{i\in I}$ to the ideal functionality on behalf of the parties in I, and receives back the output y.

– For simulating P_h during the decryption round (round 3), the simulator computes the evaluated ciphertexts $c_j^* = (\mathbf{v}_j, w_j, D)$ for each output bit j as per the protocol. Moreover, it knows the extracted values $\{\mathbf{s}_D^i\}_{i\in I}$ (see previous bullet) for dishonest parties and the values $\{\mathbf{s}_D^k\}_{k\notin I, k\neq h}$ chosen on behalf of honest parties. It then computes the decryption shares on behalf of P_h for each output bit j as:

$$w_j^h := w_j - \langle \mathbf{v}_j, \sum_{\ell\in[N]\setminus\{h\}} \mathbf{s}_D^\ell \rangle - y[j] + 2\cdot e_j$$

where $e_j \xleftarrow{\$} [-B_{smdg}^{dec}, B_{smdg}^{dec}]$ is chosen at random and $y[j]$ is the jth bit of the output y. It send the values $w_{h,j}$ on behalf of P_h.

Hybrid Games. We now define a series of *hybrid games* that will be used to prove the indistinguishability of the real and ideal worlds:

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \stackrel{\text{c}}{\equiv} \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \tag{1}$$

The output of each game is always just the output of the environment.

The game $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$**:** This is exactly an execution of the protocol π in the real world with environment $\mathcal{Z}$ and semi-mlaicious adversary $\mathcal{A}$.

The game $\text{HYB}^1_{\pi,\mathcal{A},\mathcal{Z}}$**:** In this game, we modify the real world experiment as follows. Assume (as a mental experiment) that P_h is given all the secret keys $\mathbf{s}_D^1, \ldots, \mathbf{s}_D^N$ and (as written on the "witness tape" of the adversary in round 2 or chosen by honest parties), and the ideal output y as in the simulation. Then, during the decryption protocol, it computes the decryption shares as the simulator via:

$$w_j^h := w_j - \langle \mathbf{v}_j, \sum_{\ell\in[N]\setminus\{h\}} \mathbf{s}_D^\ell \rangle - y[j] + 2\cdot e_j$$

for the output bit j (instead of using its own secret share $\mathbf{s}_D^h$). Moreover, we define the output of the honest parties in this game to be the ideal output y rather than the result of the real computation.

The game $\text{HYB}^{2.(d_1,d_2)}_{\pi,\mathcal{A},\mathcal{Z}}$**:** This game is just like an execution of HYB^1 with the following differences. In round 1, for everly level $d > d_1$ it chooses the values $\{b_{d,i,\tau}^{\ell,h}\}_{i,\tau,\ell}$ uniformly at random instead of compting them as specified. In round 2, for every level $d > d_2$, P_h chooses the values $\{\beta_{d,i,j,\tau}^{\ell,h}\}_{i,j,\tau,\ell}$ uniformly at random instead of as specified.

The game $\text{HYB}^3_{\pi,\mathcal{A},\mathcal{Z}}$**:** This is the same as $\text{HYB}^{2.(0,0)}$ with the modification that now *every* honest party P_k also just sends encryptions of 0 in round 2 instead of encrypting the bits corresponding to its real inputs $\mathbf{x}_k$.

The game $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$**:** This is the ideal-world execution with simulator $\mathcal{S}$ and environment $\mathcal{Z}$.

Claim C.3. *It holds that:* $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \stackrel{\text{s}}{\equiv} \text{HYB}^{1}_{\pi,\mathcal{A},\mathcal{Z}}$.

Proof: Firstly, we can assume the "witnesses" that $\mathcal{A}$ writes on its auxiliary witness tape on behalf of a corrupted parties $\{P_k \ : \ k \in I\}$ in various rounds contains the same *consistent* values $\mathbf{s}_d^k$ in each round. This follows by the uniqueness of secret-keys (Claim 2.3) which implies that the values $\mathbf{p}_d^k$ sent on behalf of P_k in the first round already *information theoretically commit* the attacker $\mathcal{A}$ to unique corresponding witness $\mathbf{s}_d^k$. Therefore we will speak of well-defined, consistent values $\mathbf{s}_d^k$ used by the adversary $\mathcal{A}$ on behalf of P_k, independently of which round we're talking about.

The view of the adversary is exactly the same in both worlds, except for the message that P_h sends in the last round and the output y. In HYB^1, P_h sends the "decryption shares"

$$w_j^h := w_j - \langle \mathbf{v}_j, \sum_{\ell \neq h} \mathbf{s}_D^\ell \rangle - y_j + 2 \cdot e_j$$

for each output bit j, where $e_j \stackrel{\$}{\leftarrow} [-B^{dec}_{smdg}, B^{dec}_{smdg}]$ and y_j is the jth bit of the ideal output y. By correctness (Theorem C.1), we know that $(\mathbf{v}_j, w_j)$ actually encrypts y_j with noise at most ρ so that

$$w_j = \langle \mathbf{v}_j, \sum_{\ell} \mathbf{s}_D^\ell \ \rangle + 2 \cdot e'_j + y_j$$

for some noise $e'_j \in [-\rho, \rho]$. Therefore, combining the above two equations, we can write

$$w_j^h = w_j - \langle \mathbf{v}_j, \sum_{\ell \neq h} \mathbf{s}_D^\ell \rangle - y_j + 2 \cdot e_j = \langle \mathbf{v}_j, \mathbf{s}_D^h \rangle + 2(e'_j + e_j)$$

This matches the *real world* protocol, up to the "error" being $e'_j + e_j$ rather than just e_j. However, since $\rho / B^{dec}_{smdg} = \mathsf{negl}(\kappa)$, we can apply the "smudging lemma" (Lemma 2.1) to see that this is statistically indistinguishable.

■

Claim C.4. *It holds that* $\text{HYB}^{1}_{\pi,\mathcal{A},\mathcal{Z}} \equiv \text{HYB}^{2.(D,D)}_{\pi,\mathcal{A},\mathcal{Z}}$ *and, for all* $d \in [D]$,

$$\text{HYB}^{2.(d,d)}_{\pi,\mathcal{A},\mathcal{Z}} \stackrel{\text{c}}{\equiv} \text{HYB}^{2.(d-1,d)}_{\pi,\mathcal{A},\mathcal{Z}} \stackrel{\text{c}}{\equiv} \text{HYB}^{2.(d-1,d-1)}_{\pi,\mathcal{A},\mathcal{Z}}.$$

Proof: The equivalence of $\text{HYB}^1 \equiv \text{HYB}^{2.(D,D)}$ just follows directly from the definition.

Next, let us show $\text{HYB}^{2.(d,d)} \stackrel{\text{c}}{\equiv} \text{HYB}^{2.(d-1,d)}$. The only difference between the two games is in the values $\{b^{\ell,h}_{d,i,\tau}\}_{i,\tau,\ell}$ sent on behalf of P_h in round 1. Notice that, in both of the games, the secret $\mathbf{s}_d^h$ is *only* used to compute the values $\{b^{\ell,h}_{d,i,\tau}\}_{i,\tau,\ell}$ and $\mathbf{p}_d^h$ *and nothing else.* Therefore, we can prove indistinguishability of the two games via two applications of the $\text{LWE}_{n,q_d,\varphi,\chi}$ assumption: first we apply it to switch all of $\{b^{\ell,h}_{d,i,\tau}\}$ *and* $\mathbf{p}_d^h$ from being chosen honestly to being uniformly random, and next we apply it again to switch $\mathbf{p}_d^h$ back to being chosen honestly.

Finally, let us show $\text{HYB}^{2.(d-1,d)} \stackrel{\text{c}}{\equiv} \text{HYB}^{2.(d-1,d-1)}$. The only difference between the two games is in the messages $\{\boldsymbol{\alpha}^{\ell,h}_{d,i,j,\tau}, \beta^{\ell,h}_{d,i,j,\tau}\}_{i,j,\tau,\ell}$ sent on behalf of the honest party P_h in round 2. In $\text{HYB}^{2.(d,d)}$, these values are computed in a complicated way, but are "blinded" by $\mathsf{E.PubEnc}_{\mathbf{p}_d^*}(0) + (\mathbf{0}, 2e)$ where

$e \xleftarrow{\$} [-B^{eval}_{smdg}, B^{eval}_{smdg}]$, while in $\text{HYB}^{2.(d-1,d-1)}$, they are chosen uniformly at random. Note that, in both of the hybrid games, the secret key $\mathbf{s}^h_d$ is *only used* to compute $\mathbf{p}^h_d$ and nothing else. Therefore, we only need to show that the values $\mathsf{E.PubEnc}_{\mathbf{p}^*_d}(0) + (\mathbf{0}, 2e)$ are indistinguishable from uniform. Unfortunately, the public key $\mathbf{p}^*_d$ is not chosen honestly and is partially under adversarial control. Nevertheless, we can show indistinguishability of each such value (one-at-a-time) via the *security of joint key* (Lemma 3.4).

The reduction $\mathcal{A}'$ plays $\mathsf{JoinKeys}_{\mathcal{A}'}(\mathsf{params})$. It gets a challenge public key $pk = (A, \mathbf{p})$ which it plugs into the setup matrix $A_d := A$ and the value $\mathbf{p}^h_d := \mathbf{p}$ sent on behalf of P_h in round 1 of the protocol. It also chooses $\mathbf{p}^k_d = A_d \cdot \mathbf{s}_d + 2\mathbf{e}^k_d$ at random for honest parties $k \neq h, k \notin I$ and gets the values $\mathbf{p}^k_d = A_d \cdot \mathbf{s}^k_d + 2\mathbf{e}^k_d$ and $\mathbf{s}^k_d, \mathbf{e}^k_d$ on behalf of corrupted parties $k \in I$ from the "witness tape" of the semi-malicious attacker at the end of round 1, with the guarantee that $\mathbf{e}^k_d \in [-B_\chi, B_\chi]^m$. It then sets $\mathbf{p}' = \sum_{k \neq h} \mathbf{p}^k_d, \mathbf{s}' = \sum_{k \neq h} \mathbf{s}^k_d, \mathbf{e}' = \sum_{k \neq h} \mathbf{e}^k_d$ and gives $(\mathbf{p}', \mathbf{s}', \mathbf{e}')$ to its challenger and chooses the challenge message $\mu = 0$. Finally it gets a challenge ciphertext from the challenge which it uses as the "blind" when computing $(\boldsymbol{\alpha}^{\ell,h}_{d,i,j,\tau}, \beta^{\ell,h}_{d,i,j,\tau})$. If the challenger runs with bit $\beta = 0$ (e.g. random ciphertexts) than this perfectly simulates $\text{HYB}^{2.(d-1,d)}$ and else (e.g. honest ciphertext with smudging noise B^{eval}_{smdg}) it perfectly simulates $\text{HYB}^{2.(d-1,d-1)}$. Therefore, we get indistinguishability as long at the requirements of Lemma 3.4 are satisfied, which occurs when we set $B^{eval}_{smdg} = 2^{\omega(\log(\kappa))} \cdot B_\chi$. ■

Claim C.5. *It holds that:* $\text{HYB}^{2.(0,0)}_{\pi,\mathcal{A},\mathcal{Z}} \stackrel{c}{\equiv} \text{HYB}^{3}_{\pi,\mathcal{A},\mathcal{Z}}$.

Proof: Note that, in both games, the only infomration given about the secret $\mathbf{s}^h_0$ of party P_h is just the public-key component $\mathbf{p}^h_0$. Again, we only need to prove that ciphertexts encrypted under the joint public key $pk = (A_0, \mathbf{p}^*_0)$ are "semantically secure" so that the attacker cannot distinguish encryptions of the correct input bits from encryptions of 0. To show this, we use the *security of joint key* in the exact same way as in the proof of Claim C.4 (second part). We use it twice: first to switch from encryptions of the correct input bits to random ciphertexts and then again to switch from random ciphertexts to encryptions of 0. To apply security of joint keys, we use the fact that $B^{enc}_{smdg} = 2^{\omega(\log(\kappa))} \cdot B_\chi$. ■

Finally, we notice that HYB^3 is the ideal world simulation. Therefore, equation 1 follows by combining Claims C.3, C.4 and C.5. We conclude that:

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \stackrel{c}{\equiv} \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$$

Handling abort. We now consider the case where an adversary $\mathcal{A}$ may abort the execution. In this case, the simulator $\mathcal{S}$ works exactly the same with the only difference that it sends abort to the trusted party in case $\mathcal{A}$ aborts the (simulated) execution. In case $\mathcal{A}$ does not abort, $\mathcal{S}$ sends $\mathsf{continue}$ to the trusted party at the end of the execution. We now turn to show that the ensembles are indistinguishable. We have already seen it for the case that the adversary does not abort. In case the adversary aborts, both in the ideal and real executions, the honest parties output $\perp$, and so we need to consider only the view of the adversary. However, we have already seen that the whole view is indistinguishable, and so any partial view is indistinguishable as well. This completes the proof of Theorem 5.2. ■

D Generalized Functionalities

Our basic MPC construction from Section 5 only considers deterministic functionalities where all the parties receive the same output. We would like to generalize it to handle with randomized functionalities and individual outputs. First, the standard transformation from a randomized functionality to a deterministic one (See [Gol04], Section 7.3) works for this case as well. In this transformation, instead of computing some randomized function $g(x_1, \ldots, x_N; r)$, the parties compute the deterministic function $f((r_1, x_1), \ldots, (r_N, x_N)) \stackrel{\text{def}}{=} g(x_1, \ldots, x_N; \oplus_{i=1}^{N} r_i)$. We note that this computation does not add any additional round.

Next, we move to individual outputs. Again, we use a standard transformation (See [LP09], for example). Given a function $g(x_1, \ldots, x_N) \mapsto (y_1, \ldots, y_N)$, the parties can evaluate the following function which has a single output:

$$f\left((k_1, x_1), \ldots, (k_N, x_N)\right) = \left(g_1(x_1, \ldots, x_N) \oplus k_1 \;||\; \ldots \;||\; g_N(x_1, \ldots, x_N) \oplus k_N\right)$$

where $a||b$ denotes a concatenation of a with b, and g_i indicates the ith output of g. Then, the parties can evaluate f which is a single output functionalities, instead of g. The only difference is that f has one additional exclusive-or gate for every circuit-output wire. Again, this transformation does not add any additional round of interaction.

Alternatively, for the case of individual outputs, we can just modify our protocol so that, during the decryption round (Round III), the parties will send their "decryption shares" of each output bit over a private channel to the appropriate party for whom the output bit is intended, instead of just using broadcast.

We conclude:

Theorem D.1. *Let $f : (\{0,1\}^{\ell_{in}})^N \to (\{0,1\}^{\ell_{out}})^N$ be any (possibly randomized) poly-time function. Under the same conditions as Theorem 5.2, there exists a protocol π_f that securely UC-realizes the ideal functionality $\mathcal{F}_f$ in the presence of a static* semi-malicious *adversary (Definition A.2), corrupting any $t \leq N$ parties.*

E From Semi to Fully Malicious

E.1 The Zero-Knowledge Functionality

We start with a formal description of the zero-knowledge functionality. In this case, we have one prover, P_1, and several verifiers, $P_2, \ldots, P_N$.

FUNCTIONALITY E.1 (The Zero-Knowledge Functionality $\boldsymbol{\mathcal{F}_{\mathsf{zk}}^{R}}$)**.**

- The functionality is parameterized with an NP relation R of an NP language L.
- Upon receiving a common input (x, sid) from $P_1, \ldots, P_N$, and an input (w, sid) from P_1, check that $R(x, w) = 1$. If so send $(1, sid)$ to all parties.

E.2 From Semi-Malicious to Malicious

In this section we present a general transformation from the semi-malicious model to the malicious model. Namely, given a protocol π in the semi-malicious model in the authenticated broadcast

channel model, we present a protocol π' in the $\mathcal{F}_{\mathsf{zk}}$-hybrid model, that is secure in the presence of a malicious adversary (in the broadcast channel model as well). In the CRS mode, we can use UC-NIZKs [SCO+01] to obtain a 3-round UC-secure protocol.

We denote by $\mathsf{NextMessage}_\ell^k(x_k; r_k; m_1, \ldots, m_{\ell-1})$ the next message function of party P_k in round ℓ. This is a deterministic function that takes the input x_k of party P_k, its random tape r_k, and the messages it has received so far $(m_1, \ldots, m_{\ell-1})$. The function outputs the message of P_k in round ℓ according to the protocol π. Giving a protocol π in the broadcast communication model, we parse the code of the protocol π as the sequence of $\{\mathsf{NextMessage}_\ell^k(x_k; r_k; m_1, \ldots, m_{\ell-1})\}_{k,\ell}$.

In order to verify that indeed each party follows the description of the protocol, at each round the party has to prove in zero knowledge that the message that it has produced is consistent. Formally, let $L_{k,\ell} = \{(m_1, \ldots, m_\ell - 1, m_\ell^k)\}$ be the language that consists of all the messages that were broadcasts in levels $1, \ldots, \ell - 1$, and the message m_ℓ^k that was broadcasted by P_k in round ℓ. Let $R_{k,\ell}$ be the relation that gets as input $x = (m_1, \ldots, m_{\ell-1}, m_\ell^k)$ and a witness $w = (x_k, r_k)$, and returns 1 if and only if $\mathsf{NextMessage}_\ell^k(x_k, r_k, m_1, \ldots, m_{\ell-1}) = m_\ell^k$. In addition, let $r(\pi)$ denote the number of rounds in π.

In case a party abort, we replace its message with the special symbol $\perp$. The next message function $\mathsf{NextMessage}$ should specify how to react in such a case.

PROTOCOL E.2 (The protocol π' in the f_{zk}-hybrid model)**.**

- **Input:** An input x_k, and a random tape r_k.
- **Common input:** The semi-malicious protocol π which is parsed as $\{\mathsf{NextMessage}_\ell^k(x_k; r_k; m_1, \ldots, m_{\ell-1})\}_{k,\ell}$.
- **The protocol:**
  1. For each round $\ell = 1, \ldots, r(\pi)$:
     (a) Giving all the messages $m_{\ell-1}^1, \ldots, m_{\ell-1}^N$ that were broadcasted in round $\ell - 1$, concatenate them into m_ℓ, and store it.
     (b) Compute $m_\ell^k = \mathsf{NextMessage}_\ell^k(x_k, r_k, m_1, \ldots, m_{\ell-1})$. Broadcast m_ℓ^k.
     (c) For any $k' = 1, \ldots, N$:
     The parties invoke the $\mathcal{F}_{\mathsf{zk}}^{R_{k',\ell}}$ on a common input $(m_1, \ldots, m_{\ell-1}, m_\ell^{k'})$. In addition, for $k = k'$, P_k acts as the prover and inserts its private input $w = (x_k, r_k)$.
  2. Output $\mathsf{NextMessage}_{r(\pi)}^k(x_k; r_k; m_1, \ldots, m_{r(\pi)-1})$.

Theorem E.3. *Let $\mathcal{F}$ be any N-ary functionality, let π be a N-party protocol, and let $t \leq N$. If π (fairly) t-securely realizes $\mathcal{F}$ in the semi-malicious model in the authenticated broadcast channel model, then π' (fairly) t-secure in the $\mathcal{F}_{\mathsf{zk}}$-hybrid model, in the presence of a malicious adversary.*

Proof: Let $\mathcal{A}^{\mathsf{mal}}$ be a malicious adversary for π' controlling subset $I \subseteq [n]$ of cardinality at most t. We build a semi-malicious adversary $\mathcal{A}^{\mathsf{sm}}$ for π as follows. The adversary $\mathcal{A}^{\mathsf{sm}}$ is interacting with honest parties in execution of the protocol π, and uses the adversary $\mathcal{A}^{\mathsf{mal}}$ to determine the behavior of the corrupted parties. Whenever the adversary $\mathcal{A}^{\mathsf{sm}}$ receives a broadcast message from honest parties, it proceed the message to $\mathcal{A}^{\mathsf{mal}}$. At the end of each round, it simulates the invocations of the $\mathcal{F}_{\mathsf{zk}}$ functionality. When an honest party should be the prover, it just checks that the adversary sends the correct statement and returns 1 as the response of $\mathcal{F}_{\mathsf{zk}}$. In case where a corrupted party is the prover, it check the statement and the witness (the input and the randomness). Formally,

for P_i, $i \in I$ in round ℓ, it checks that indeed $\mathsf{NextMessage}^i_\ell(x_i; r_i; m_1, \ldots, m_{\ell-1}) = m^i_\ell$. In case the above holds, it return 1 to $\mathcal{A}^{\mathsf{mal}}$, and updates the special random tape of $\mathcal{A}^{\mathsf{sm}}$ to include (x_i, r_i). In case $\mathcal{A}^{\mathsf{mal}}$ the above check fails, it aborts the party P_i. At the end of the computation, $\mathcal{A}^{\mathsf{sm}}$ outputs whatever $\mathcal{A}^{\mathsf{mal}}$ outputs and halts.

Given the adversary $\mathcal{A}^{\mathsf{sm}}$, we have a simulator $\mathcal{S}^{\mathsf{sm}}$, for which for every environment $\mathcal{Z}$, the ideal and real are indistinguishable. We build a simulator $\mathcal{S}^{\mathsf{mal}}$ using $\mathcal{S}^{\mathsf{sm}}$. The simulator works exactly the same. This include querying the functionality as $\mathcal{S}^{\mathsf{sm}}$, aborting in case $\mathcal{S}^{\mathsf{sm}}$ aborts (if they both allowed to abort), and interacting with the environment in the same way. Since $\mathcal{S}^{\mathsf{mal}}$ and $\mathcal{S}^{\mathsf{sm}}$ works exactly the same, and since $\mathcal{A}^{\mathsf{mal}}$ and $\mathcal{A}^{\mathsf{sm}}$ has the same behavior, we have that for every environment $\mathcal{Z}$:

$$\mathrm{HYB}^{\mathcal{F}_{\mathsf{zk}}}_{\mathcal{F},\mathcal{S}^{\mathsf{mal}},\mathcal{Z}} \stackrel{\mathrm{c}}{\equiv} \mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$$

■

F Efficient Σ-Protocols and Fully-Malicious Compiler

In this section, we give an efficient compiler for transforming our MPC protocol for semi-malicious adversaries into one that is secure against fully malicious adversaries. In order to achieve this objective, we will construct efficient NIZKs in the random oracle model and then use them to compile the semi-malicious protocol. We start by constructing gap Σ-protocols for some LWE-based languages. We will then show how to use these Σ-protocols for the "required" relations by using a series of AND and OR proofs. Finally, we can compile the Σ-protocols into efficient UC NIZKs in the random oracle model, and then use these NIZKs to compile the semi-malicious protocol and obtain an efficient 3-round MPC protocol against fully malicious adversaries.

The rest of this section is organized as follows. We start by recalling the syntax and security properties of gap Σ-protocols in Section F.1. Next, in Sections F.2 and F.3, we construct gap Σ-protocols for some LWE-based languages. Finally, in Section F.4, we show how to use the Σ-protocols (from Sections F.2 and F.3) for the required relations in the semi-malicious protocol.

F.1 Gap Σ-Protocols

We consider a weaker version of Σ-protocols [CDS94] that we call *gap* Σ-protocols. Let $\mathcal{R}_{\mathsf{zk}}, \mathcal{R}_{\mathsf{sound}} \subseteq \{0,1\}^* \times \{0,1\}^*$ be two NP relations such that $\mathcal{R}_{\mathsf{zk}} \subseteq \mathcal{R}_{\mathsf{sound}}$, and $L_{\mathsf{zk}} = \{x : \exists\ w \text{ s.t. } (x,w) \in \mathcal{R}_{\mathsf{zk}}\}$, $L_{\mathsf{sound}} = \{x : \exists\ w \text{ s.t. } (x,w) \in \mathcal{R}_{\mathsf{sound}}\}$ be the corresponding languages. A gap Σ-protocol $\langle P, V\rangle$ for $(\mathcal{R}_{\mathsf{zk}}, \mathcal{R}_{\mathsf{sound}})$ is a three-move interactive protocol between a prover P and a verifier V of the following form. P and V get as common input a statement $x \in L_{\mathsf{zk}}$, additionally P gets as private input $w \in \mathcal{R}_{\mathsf{zk}}(x)$. $\langle P, V\rangle$ proceeds as follows:

- P samples a value a and sends it to V.
- V samples a random "challenge" $c \xleftarrow{\$} \Phi$ from the "challenge space" Φ and sends it to P.
- P gives an answer z. V either accepts or rejects the conversation (a, c, z) for statement x.

A gap Σ-protocol must satisfy the following properties:

Correctness: For any $(x,w) \in \mathcal{R}_{\mathsf{zk}}$, $V(x)$ always accepts the interaction with $P(x,w)$.

Special Soundness: There exists a PPT extractor E such that for any two valid transcripts (a, c, z), (a, c', z') for a statement x, if $c \neq c'$ and $w \leftarrow E(x, a, c, z, c', z')$ then $w \in \mathcal{R}_{\mathsf{sound}}(x)$.

Honest-Verifier Zero-Knowledge: There exists a PPT simulator S, such that for any statement $x \in L_{\mathsf{zk}}$ and challenge $c \in \Phi$ we have $(a', c, z') \stackrel{s}{\equiv} (a, c, z)$, where $(a', z') \leftarrow S(x, c)$ and (a, c, z) denotes a protocol transcript between $P(x, w)$ and $V(x)$ with challenge c (for $w \in \mathcal{R}_{\mathsf{zk}}(x)$).

F.2 $\langle P, V\rangle_{\mathsf{lwe}}$: A Gap Σ-Protocol for LWE

In this section, we give a gap Σ-protocol $\langle P, V\rangle_{\mathsf{lwe}}$ that allows a prover P to prove to a verifier V that a given tuple $\mathbf{A}, \mathbf{b}$ is an LWE tuple; i.e., there exists a secret vector $\mathbf{s}$ and a "short" error vector $\mathbf{e}$ such that $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + 2\mathbf{e}$.

More formally, let κ be the security parameter, and $q = q(\kappa)$ be an odd integer modulus. In the discussion below, unless stated otherwise, all the operations are performed mod q. Let $m = m(\kappa)$, $n = n(\kappa)$, and $B = B(\kappa)$ be positive integers. We define the NP relations:

$$\mathcal{R}^B_{\mathsf{lwe}} \;=\; \{(\mathbf{A}, \mathbf{b}),\ (\mathbf{s}, 2\mathbf{e}) \;:\; \mathbf{b} = \mathbf{A} \cdot \mathbf{s} + 2\mathbf{e},\ \mathbf{A} \in \mathbb{Z}_q^{m \times n},\ \mathbf{b} \in \mathbb{Z}_q^m,\ \mathbf{s} \in \mathbb{Z}_q^n,\ 2\mathbf{e} \in [-B, B]^m\}.$$

Let L^B_{lwe} be the NP language corresponding to the relation $\mathcal{R}^B_{\mathsf{lwe}}$. For integers $B^* \geq B$ we have $\mathcal{R}^B_{\mathsf{lwe}} \subseteq \mathcal{R}^{B^*}_{\mathsf{lwe}}$ and $L^B_{\mathsf{lwe}} \subseteq L^{B^*}_{\mathsf{lwe}}$.

Protocol $\langle P, V\rangle_{\mathsf{lwe}}$. We now give a gap Σ-protocol $\langle P, V\rangle_{\mathsf{lwe}}$ for the relation pair $(\mathcal{R}^B_{\mathsf{lwe}}, \mathcal{R}^{B^*}_{\mathsf{lwe}})$, which is secure as long as $B/B^* \leq \mathsf{negl}(\kappa)$. The machines P and V get as common input the statement $(\mathbf{A}, \mathbf{b}) \in L^B_{\mathsf{lwe}}$. Additionally, P gets as private input a witness tuple $(\mathbf{s}, 2\mathbf{e}) \in \mathcal{R}^B_{\mathsf{lwe}}(\mathbf{A}, \mathbf{b})$. Protocol $\langle P, V\rangle_{\mathsf{lwe}}$ proceeds as follows:

1. P samples a random vector $\mathbf{s}' \stackrel{\$}{\leftarrow} \mathbb{Z}_q^n$ and an "even" error vector $2\mathbf{e}' \stackrel{\$}{\leftarrow} [-(\frac{B^*}{2} - B), (\frac{B^*}{2} - B)]^m$. It then sends $\mathbf{b}' = \mathbf{A} \cdot \mathbf{s}' + 2\mathbf{e}'$ to V.
2. V sends a random challenge $c \in \{0, 1\}$ to P.
3. P sends $\mathbf{z} = \mathbf{s}' + c\mathbf{s}$ to V.
4. V outputs 1 iff $(\mathbf{b}' + c\mathbf{b} - \mathbf{A} \cdot \mathbf{z})$ is an "even" noise vector $2\mathbf{e}^* \in [-\frac{B^*}{2}, \frac{B^*}{2}]^m$.

Theorem F.1. *Protocol $\langle P, V\rangle_{\mathsf{lwe}}$ is a gap Σ-protocol for the relation pair $(\mathcal{R}^B_{\mathsf{lwe}}, \mathcal{R}^{B^*}_{\mathsf{lwe}})$ as long as $B/B^* \leq \mathsf{negl}(\kappa)$.*

Proof: We will prove that $\langle P, V\rangle_{\mathsf{lwe}}$ satisfies each of the three properties – completeness, special soundness, and honest verifier zero-knowledge.

Completeness. If P is honest, we have

$$\mathbf{b}' + c\mathbf{b} - \mathbf{A} \cdot \mathbf{z} = \mathbf{A} \cdot \mathbf{s}' + 2\mathbf{e}' + c(\mathbf{A} \cdot \mathbf{s} + 2\mathbf{e}) - \mathbf{A} \cdot (\mathbf{s}' + c\mathbf{s}) = 2\mathbf{e}' + 2c\mathbf{e} \in \left[-\frac{B^*}{2}, \frac{B^*}{2}\right]^m .$$

Further, $2\mathbf{e}' + 2c\mathbf{e}$ is clearly "even".

Special Soundness. Let $(\mathbf{b}', 1, \mathbf{z}_1)$, $(\mathbf{b}', 0, \mathbf{z}_2)$ be two transcripts for protocol $\langle P, V\rangle_{\mathsf{lwe}}$. The extractor $E(\mathbf{A}, \mathbf{b}, \mathbf{b}', 1, \mathbf{z}_1, 0, \mathbf{z}_2)$ simply outputs $(\mathbf{s}^*, 2\mathbf{e}^*)$ defined as:

$$\mathbf{s}^* := \mathbf{z}_1 - \mathbf{z}_2 \quad , \quad 2\mathbf{e}^* := \mathbf{b} - \mathbf{A}(\mathbf{z}_1 - \mathbf{z}_2).$$

If both transcripts are *valid*, then there exist "even" noise vectors $2\mathbf{e}_1^*, 2\mathbf{e}_2^* \in [-\frac{B^*}{2}, \frac{B^*}{2}]^m$ such that:

$$\mathbf{b}' + \mathbf{b} - \mathbf{A} \cdot \mathbf{z}_1 = 2\mathbf{e}_1^*, \quad \mathbf{b}' - \mathbf{A} \cdot \mathbf{z}_2 = 2\mathbf{e}_2^*.$$

Subtracting the above equations, we get $\mathbf{b} = \mathbf{A}(\mathbf{z}_1 - \mathbf{z}_2) + 2(\mathbf{e}_1^* - \mathbf{e}_2^*)$. Further, note that $2\mathbf{e}^* = 2\mathbf{e}_1^* - 2\mathbf{e}_2^* \in [-B^*, B^*]$. It follows immediately that $(\mathbf{s}^*, 2\mathbf{e}^*) \in \mathcal{R}_{\mathsf{lwe}}^{B^*}(\mathbf{A}, \mathbf{b})$.

Honest Verifier Zero Knowledge. We first give the simulator strategy $S(\mathbf{A}, \mathbf{b}, c)$:

- Sample $\mathbf{z} \xleftarrow{\$} \mathbb{Z}_q^n$, $2\mathbf{e}^* \xleftarrow{\$} [-(\frac{B^*}{2} - B), (\frac{B^*}{2} - B)]^m$.
- Compute $\mathbf{b}' = \mathbf{A} \cdot \mathbf{z} + 2\mathbf{e}^* - c\mathbf{b}$.
- Output $(\mathbf{b}', c, \mathbf{z})$.

First note that $\mathbf{b}' + c\mathbf{b} - \mathbf{A} \cdot \mathbf{z} = 2\mathbf{e}^* \in [-\frac{B^*}{2}, \frac{B^*}{2}]^m$, and clearly $2\mathbf{e}^*$ is "even". Thus, the above transcript is accepting. Further note that when $(\mathbf{A}, \mathbf{b}) \in L_{\mathsf{lwe}}^B$, there exists $\mathbf{s} \in \mathbb{Z}_q^n, \mathbf{e} \in [-B, B]^m$ such that $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + 2\mathbf{e}$. Thus, in the simulated transcript $(\mathbf{b}', c, \mathbf{z})$, we have $\mathbf{b}' = \mathbf{A} \cdot \mathbf{z} + 2\mathbf{e}^* - c\mathbf{b} = \mathbf{A} \cdot \mathbf{s}' + 2\mathbf{e}'$, where $\mathbf{s}' = \mathbf{z} - c\mathbf{s} \in \mathbb{Z}_q^n$, and $2\mathbf{e}' = 2\mathbf{e}^* - 2c\mathbf{e}$. When $c = 0$, it is straightforward to see the distribution of the transcript output by S (over the coins of S) is identical to the distribution of the transcript from honest execution between P and V. When $c = 1$, the only difference between the honestly generated transcript and a simulated one is that instead of using "even" noise $2\mathbf{e}' \xleftarrow{\$} [-(\frac{B^*}{2} - B), (\frac{B^*}{2} - B)]^m$, the "effective" noise used in the computation of $\mathbf{b}'$ by S is of the form $2\mathbf{e}' - 2\mathbf{e}$, where $2\mathbf{e} \in [-B, B]^m$ is fixed. Then, since $\frac{B}{B^*/2 - B} \leq \mathsf{negl}(\kappa)$, it follows from the "smudging lemma" (Lemma 2.1) that the simulated transcript is statistically indistinguishable from a real transcript. ■

F.2.1 Applications of $\langle P, V\rangle_{\mathsf{lwe}}$

We now discuss some applications of our gap Σ-protocol $\langle P, V\rangle_{\mathsf{lwe}}$; specifically, we consider some more complicated NP relations and show that $\langle P, V\rangle_{\mathsf{lwe}}$ can be used for these relations as well. Looking ahead, these relations will be relevant in Section F.4, where we build an efficient compiler for our semi-malicious MPC protocol to achieve security against fully malicious adversaries.

Σ-protocols for "approximate" symmetric key encryptions. We show that $\langle P, V\rangle_{\mathsf{lwe}}$ can be used to prove that a given tuple $(\mathbf{a}, b)$ is an approximate encryption of a message μ under the secret key $\mathbf{s}$ corresponding to a public key $(\mathbf{A}, \mathbf{p})$. To this end, we first consider a relation pair $(\mathcal{R}_{\mathsf{approx}}^B, \mathcal{R}_{\mathsf{approx}}^{B^*})$, where $\mathcal{R}_{\mathsf{approx}}^B$ is defined as:

$$\begin{aligned}\mathcal{R}_{\mathsf{approx}}^B \quad = \quad & \{(\mathbf{a}, b, \mu, \mathbf{A}, \mathbf{p}),\ (2e, \mathbf{s}, 2\mathbf{e}) \ :\ b = \mathbf{a} \cdot \mathbf{s} + 2e + \mu,\ \mathbf{p} = \mathbf{A} \cdot \mathbf{s} + 2\mathbf{e} \\ & \mathbf{a} \in \mathbb{Z}_q^n,\ b \in \mathbb{Z}_q,\ \mu \in \mathbb{Z}_q, \mathbf{A} \in \mathbb{Z}_q^{m \times n},\ \mathbf{p} \in \mathbb{Z}_q^m,\ \mathbf{s} \in \mathbb{Z}_q^n,\ 2e \in [-B, B],\ 2\mathbf{e} \in [-B, B]^m\},\end{aligned}$$

and $\mathcal{R}^{B^*}_{\mathsf{approx}}$ is defined analogously, except that $2e \in [-B^*, B^*]$, and $2\mathbf{e} \in [-B^*, B^*]^m$.

Reducing $(\mathcal{R}^{B}_{\mathsf{approx}}, \mathcal{R}^{B^*}_{\mathsf{approx}})$ *to* $(\mathcal{R}^{B}_{\mathsf{lwe}}, \mathcal{R}^{B^*}_{\mathsf{lwe}})$. There is a simple transformation between

$$\mathcal{R}^{B}_{\mathsf{approx}} = \{x = (\mathbf{a}, b, \mu, \mathbf{A}, \mathbf{p}), w = (2e, \mathbf{s}, 2\mathbf{e})\} \iff \mathcal{R}^{B}_{\mathsf{lwe}} = \{x' = (\mathbf{A}', \mathbf{b}'), w' = (\mathbf{s}', 2\mathbf{e}')\}$$

by simply equating

$$\mathbf{A}' = \begin{bmatrix}\mathbf{A}\\ \mathbf{a}\end{bmatrix}, \mathbf{b}' = \begin{bmatrix}\mathbf{p}\\ b-\mu\end{bmatrix}, \mathbf{s}' = \mathbf{s}, 2\mathbf{e}' = \begin{bmatrix}2\mathbf{e}\\ 2e\end{bmatrix}.$$

This transformation ensures $(x, w) \in \mathcal{R}^{B}_{\mathsf{approx}}$ iff $(x', w') \in \mathcal{R}^{B}_{\mathsf{lwe}}$. The same idea can be used to transform an instance of relation $\mathcal{R}^{B^*}_{\mathsf{approx}}$ into an instance of relation $\mathcal{R}^{B^*}_{\mathsf{lwe}}$. One can therefore use the protocol $\langle P, V\rangle_{\mathsf{lwe}}$ directly to a gap protocol for $(\mathcal{R}^{B}_{\mathsf{approx}}, \mathcal{R}^{B^*}_{\mathsf{approx}})$.

Extension to multiple Noise Levels. We can also define a variant of the above relation where the noise level in the public key and the ciphertext differ.

$$\begin{aligned}\mathcal{R}^{B_1,B_2}_{\mathsf{approx}} \;=\; & \{(\mathbf{a}, b, \mu, \mathbf{A}, \mathbf{p}),\ (2e, \mathbf{s}, 2\mathbf{e}) \ :\ b = \mathbf{a}\cdot\mathbf{s} + 2e + \mu,\ \mathbf{p} = \mathbf{A}\cdot\mathbf{s} + 2\mathbf{e}\\ & \mathbf{a} \in \mathbb{Z}_q^n,\ b \in \mathbb{Z}_q,\ \mu \in \mathbb{Z}_q, \mathbf{A} \in \mathbb{Z}_q^{m\times n},\ \mathbf{p} \in \mathbb{Z}_q^m,\ \mathbf{s} \in \mathbb{Z}_q^n,\ 2e \in [-B_2, B_2],\ 2\mathbf{e} \in [-B_1, B_1]^m\},\end{aligned}$$

The gap Σ-protocol for this relation is a simple extension of the one above and that for $\mathcal{R}^{B}_{\mathsf{lwe}}$.

Σ-protocol for Secret Key Bits. We show that protocol $\langle P, V\rangle_{\mathsf{lwe}}$ can also be used to prove that a particular component of the secret key vector $\mathbf{s}$ corresponding to a public key $(\mathbf{A}, \mathbf{p})$ is 0 or 1. To this end, we first consider a relation pair $(\mathcal{R}^{B,i,c}_{\mathsf{bit}}, \mathcal{R}^{B^*,i,c}_{\mathsf{bit}})$, where $\mathcal{R}^{B,i,c}_{\mathsf{bit}}$ is defined as follows:

$$\mathcal{R}^{B,i,c}_{\mathsf{bit}} \;=\; \left\{(\mathbf{A}, \mathbf{p}),\ (\mathbf{s}, 2\mathbf{e}) \ :\ \begin{array}{l}\mathbf{p} = \mathbf{A}\cdot\mathbf{s} + 2\mathbf{e},\ \mathbf{s}[i] = c\\ \mathbf{A} \in \mathbb{Z}_q^{m\times n},\ \mathbf{p} \in \mathbb{Z}_q^m,\ \mathbf{s} \in \mathbb{Z}_q^n,\ 2\mathbf{e} \in [-B, B]^m\end{array}\right\},$$

and $\mathcal{R}^{B^*,i,c}_{\mathsf{bit}}$ is defined analogously, except that $2e \in [-B^*, B^*]$, and $2\mathbf{e} \in [-B^*, B^*]^m$.

Reducing $(\mathcal{R}^{B,i,c}_{\mathsf{bit}}, \mathcal{R}^{B^*,i,c}_{\mathsf{bit}})$ *to* $(\mathcal{R}^{B}_{\mathsf{lwe}}, \mathcal{R}^{B^*}_{\mathsf{lwe}})$. Let the vector $\mathbf{a}_i \in \mathbb{Z}_q^m$ denote the i^{th} column of matrix $\mathbf{A}$. Let $\mathbf{A}_{-i}$ be the matrix $\mathbf{A}$ with the i^{th} column removed. Then

$$\mathbf{p} = \mathbf{A}\cdot\mathbf{s} + 2\mathbf{e} \ \wedge\ \mathbf{s}[i] = c \quad \text{iff} \quad \mathbf{A}_{-i}\mathbf{s}[-i] + 2\mathbf{e} = \mathbf{p} - c\cdot\mathbf{a}_i.$$

where $\mathbf{s}[-i]$ denotes the vector $\mathbf{s}$ with the i^{th} component removed. Therefore, there is a simple transformation between

$$\mathcal{R}^{B,i,c}_{\mathsf{bit}} = \{x = (\mathbf{A}, \mathbf{p}), w = (\mathbf{s}, 2\mathbf{e})\} \iff \mathcal{R}^{B}_{\mathsf{lwe}} = \{x' = (\mathbf{A}', \mathbf{b}'), w' = (\mathbf{s}', 2\mathbf{e}')\}$$

given by equating

$$\mathbf{A}' = \mathbf{A}_{-i},\quad \mathbf{b}' = \mathbf{p} - c\mathbf{a}_i,\quad \mathbf{s}' = \mathbf{s}[-i],\quad 2\mathbf{e} = 2\mathbf{e}'.$$

This transformation ensures $(x, w) \in \mathcal{R}^{B,i,c}_{\mathsf{bit}}$ iff $(x', w') \in \mathcal{R}^{B}_{\mathsf{lwe}}$. Hence the protocol $\langle P, V\rangle_{\mathsf{lwe}}$ can be used as a gap Σ-protocol for $(\mathcal{R}^{B,i,c}_{\mathsf{bit}}, \mathcal{R}^{B^*,i,c}_{\mathsf{bit}})$.

F.3 $\langle P, V\rangle_{\mathsf{enc}}$: A Σ-Protocol for LWE-based Public-Key Encryption

We now give a gap Σ-protocol $\langle P, V\rangle_{\mathsf{enc}}$ that allows a prover P to prove to a verifier V that a given tuple $(\mathbf{A}, \mathbf{p}, \mathbf{v}, w)$ is a "smudged" encryption of 0 with public key $pk = (\mathbf{A}, \mathbf{p})$; i.e., there exists a noise vector $\mathbf{r}$ and "even" noise $2e$ such that $\mathbf{v} = \mathbf{r}^T \cdot \mathbf{A}$, and $w = r \cdot \mathbf{p} + 2e$.

More formally, let κ be the security parameter, and $q = q(\kappa)$ be an odd integer modulus. Let $m = m(\kappa)$, $n = n(\kappa)$, and $B_r = B_r(\kappa)$, $B_e = B_e(\kappa)$ be positive integers. Now, we consider two NP relations $\mathcal{R}_{\mathsf{enc}}^{B_r,B_e}$ as follows:

$$\mathcal{R}_{\mathsf{enc}}^{B_r,B_e} = \big\{(\mathbf{A}, \mathbf{p}, \mathbf{v}, w),\ (\mathbf{r}, 2e)\ :\ \mathbf{v} = \mathbf{r}^T \cdot \mathbf{A},\ w = \mathbf{r} \cdot \mathbf{p} + 2e,\ \mathbf{A} \in \mathbb{Z}_q^{m\times n},\ \mathbf{p} \in \mathbb{Z}_q^m,\ \mathbf{r} \in [-B_r, B_r]^m,\ 2e \in [-B_e, B_e]\big\},$$

Let $L_{\mathsf{enc}}^{B_r,B_e}$ be the corresponding NP language. Given integers $B_r^* \geq B_r$ and $B_e^* \geq B_e$ we have $\mathcal{R}_{\mathsf{enc}}^{B_r,B_e} \subseteq \mathcal{R}_{\mathsf{enc}}^{B_r^*,B_e^*}$ and $L_{\mathsf{enc}}^{B_r,B_e} \subseteq L_{\mathsf{enc}}^{B_r^*,B_e^*}$.

Protocol $\langle P, V\rangle_{\mathsf{enc}}$. The Σ-protocol $\langle P, V\rangle_{\mathsf{enc}}$ for the relation pair $(\mathcal{R}_{\mathsf{enc}}^{B_r,B_e}, \mathcal{R}_{\mathsf{enc}}^{B_r^*,B_e^*})$ is described as follows. P and V get as common input a tuple $(\mathbf{A}, \mathbf{p}, \mathbf{v}, w) \in L_{\mathsf{enc}}^{B_r,B_e}$; additionally, P gets as private input a witness tuple $(\mathbf{r}, 2e) \in \mathcal{R}_{\mathsf{enc}}^{B_r,B_e}(\mathbf{A}, \mathbf{p}, \mathbf{v}, w)$. Protocol $\langle P, V\rangle_{\mathsf{enc}}$ proceeds as follows:

1. P samples the "short" vectors

$$\mathbf{r}' \xleftarrow{\$} \left[-(\frac{B_r^*}{2} - B_r), (\frac{B_r^*}{2} - B_r)\right]^m \quad , \quad 2e' \xleftarrow{\$} \left[-(\frac{B_e^*}{2} - B_e), (\frac{B_e^*}{2} - B_e)\right].$$

   It then computes $\mathbf{v}' = (\mathbf{r}')^T \cdot \mathbf{A}$, $w' = \mathbf{r}' \cdot \mathbf{p} + 2e'$, and sends $(\mathbf{v}', w')$ to V.

2. V sends a random challenge $c \xleftarrow{\$} \{0, 1\}$ to P.

3. P sends $\mathbf{z} = \mathbf{r}' + c\mathbf{r}$ to V.

4. Verifier V outputs 1 iff :

   - $\mathbf{z} \in [-\frac{B_r^*}{2}, \frac{B_r^*}{2}]^m$,
   - $\mathbf{z}^T \cdot \mathbf{A} = \mathbf{v}' + c\mathbf{v}$,
   - $(w' + cw - \mathbf{z} \cdot \mathbf{p})$ is an "even" noise value $2e^* \in [-\frac{B_e^*}{2}, \frac{B_e^*}{2}]$.

Theorem F.2. *Protocol $\langle P, V\rangle_{\mathsf{enc}}$ is a gap Σ-protocol for the relation pair $(\mathcal{R}_{\mathsf{enc}}^{B_r,B_e}, \mathcal{R}_{\mathsf{enc}}^{B_r^*,B_e^*})$ as long as $B_r/B_r^* \leq \mathsf{negl}(\kappa)$ and $B_e/B_e^* \leq \mathsf{negl}(\kappa)$.*

Proof: We will prove that $\langle P, V\rangle_{\mathsf{enc}}$ satisfies each of the three properties – completeness, special soundness, and honest verifier zero-knowledge.

Completeness. If P is honest, we have that

$$\mathbf{z} = \mathbf{r}' + c\mathbf{r} \in \left[-\frac{B_r^*}{2}, \frac{B_r^*}{2}\right]^m \quad , \quad \mathbf{z}^T \cdot \mathbf{A} = (\mathbf{r}' + c\mathbf{r})^T \cdot \mathbf{A} = \mathbf{v}' + c\mathbf{v}.$$

Further,

$$w' + cw - \mathbf{z} \cdot \mathbf{p} = \mathbf{r}' \cdot \mathbf{p} + 2e' + c(\mathbf{r} \cdot \mathbf{p} + 2e) - (\mathbf{r}' + c\mathbf{r}) \cdot \mathbf{p} = 2e' + 2ce \in [-\frac{B_e^*}{2}, \frac{B_e^*}{2}]$$

is small and even.

Special Soundness. Let $(\mathbf{v}, w, 1, \mathbf{z}_1)$, $(\mathbf{v}, w, 0, \mathbf{z}_2)$ be two transcripts for protocol $\langle P, V\rangle_{\mathsf{enc}}$. The extractor $E(\mathbf{A}, \mathbf{b}, \mathbf{v}, w, 1, \mathbf{z}_1, 0, \mathbf{z}_2)$ outputs $(\mathbf{s}^*, 2e^*)$ given by

$$\mathbf{r}^* = \mathbf{z}_1 - \mathbf{z}_2 \quad , \quad 2e^* = w - (\mathbf{z}_1^T - \mathbf{z}_2^T)\mathbf{A}.$$

Now, note that if both the transcripts are *valid*, the following holds:

$$\mathbf{z}_1, \mathbf{z}_2 \in [-\frac{B_r^*}{2}, \frac{B_r^*}{2}]^m, \quad \mathbf{z}_1^T \cdot \mathbf{A} = \mathbf{v}' + \mathbf{v}, \quad \mathbf{z}_2^T \cdot \mathbf{A} = \mathbf{v}', \tag{2}$$

$$\exists e_1^*, e_2^* \in [-\frac{B_e^*}{2}, \frac{B_e^*}{2}] \text{ s.t. } w' + w - \mathbf{z}_1^T \cdot \mathbf{A} = 2e_1^*, \quad w' - \mathbf{z}_2^T \cdot \mathbf{A} = 2e_2^*. \tag{3}$$

From the set of equations (2), we have that $\mathbf{v} = (\mathbf{z}_1^T - \mathbf{z}_2^T)\mathbf{A}$; also $\mathbf{r}^* = \mathbf{z}_1 - \mathbf{z}_2 \in [-B_r^*, B_r^*]$. Further, from (3), we have that $w = (\mathbf{z}_1^T - \mathbf{z}_2^T)\mathbf{A} + 2e_1^* - 2e_2^*$; also $2e^* = 2e_1^* - 2e_2^* \in [-B_e^*, B_e^*]$. Combining the above arguments, it follows immediately that $(r^*, 2e^*) \in \mathcal{R}_{\mathsf{enc}}^{B_r^*, B_e^*}(\mathbf{A}, \mathbf{p}, \mathbf{v}, w)$.

Honest Verifier Zero Knowledge. We first give the simulator strategy $S(\mathbf{A}, \mathbf{p}, \mathbf{v}, w)$:

- Sample $\mathbf{z} \xleftarrow{\$} [-(\frac{B_r^*}{2} - B_r), (\frac{B_r^*}{2} - B_r)]^m$, $2e^* \xleftarrow{\$} [-(\frac{B_e^*}{2} - B_e), (\frac{B_e^*}{2} - B_e)]$.
- Compute $\mathbf{v}' = \mathbf{z}^T \cdot \mathbf{A} - c\mathbf{v}$, $w' = \mathbf{z} \cdot \mathbf{p} + 2e^* - cw$.
- Output $(\mathbf{v}', w', c, \mathbf{z})$.

Now, first note that the transcript $(\mathbf{v}', w', c, \mathbf{z})$ computed as above is accepting because the following conditions hold: $\mathbf{z} \in [-\frac{B_r^*}{2}, \frac{B_r^*}{2}]^m$, $\mathbf{z}^T \cdot \mathbf{A} = \mathbf{v}' + c\mathbf{v}$, $w' + cw - \mathbf{z} \cdot \mathbf{p} = 2e^* \in [-\frac{B_e^*}{2}, \frac{B_e^*}{2}]$ and clearly, $2e^*$ is even.

Further note that when $(\mathbf{A}, \mathbf{p}, \mathbf{v}, w) \in L_{\mathsf{enc}}^{B_r, B_e}$, there exists $\mathbf{r} \in [-B_r, B_r]^m, e \in [-B_e, B_e]$ such that $\mathbf{v} = \mathbf{r}^T \cdot \mathbf{A}$, and $w = \mathbf{r} \cdot \mathbf{p} + 2e$. Thus, in the simulated transcript $(\mathbf{v}', w', c, \mathbf{z})$, we have $\mathbf{v}' = \mathbf{z}^T \cdot \mathbf{A} - c\mathbf{v} = (\mathbf{r}')^T \cdot \mathbf{A}$, and $w' = \mathbf{z} \cdot \mathbf{p} + 2e^* - cw = (\mathbf{r}')^T \cdot \mathbf{A} + 2e'$, where $\mathbf{r}' = \mathbf{z} - c\mathbf{r}$, and $2e' = 2e^* - 2ce$.

When $c = 0$, it is straightforward to see the distribution of the transcript output by S (over the coins of S) is identical to the distribution of the transcript from honest execution between P and V. When $c = 1$, the only differences between the honestly generated transcript and a simulated one are the following:

1. Instead of using noise vector $\mathbf{r}' \xleftarrow{\$} [-(\frac{B_r^*}{2} - B_r), (\frac{B_r^*}{2} - B_r)]^m$, the "effective" noise vector used in the computation of $\mathbf{v}'$ and w' is of the form $\mathbf{r}' - \mathbf{r}$, where $\mathbf{r} \in [-B_r, B_r]$ is fixed.

2. Further, instead of using "even" noise $2e' \xleftarrow{\$} [-(\frac{B_e^*}{2} - B_e), (\frac{B_e^*}{2} - B_e)]$, the "effective" noise value used in the computation of w' is of the form $2e' - 2e$ where $2e \in [-B, B]$ is fixed.

Then, since $\frac{B_r}{B_r^*/2 - B_r} \leq \mathsf{negl}(\kappa)$, and $\frac{B_e}{B_e^*/2 - B_e} \leq \mathsf{negl}(\kappa)$, it follows from Lemma 2.1 that the simulated transcript is statistically indistinguishable from a real transcript. ■

F.4 Efficient Compiler for Our Semi-Malicious MPC Protocol

We now give an efficient compiler for transforming our MPC protocol π (described in Section 5) that is secure against semi-malicious adversaries into one that is secure against fully malicious adversaries. Specifically, we will show how the Σ-protocols from the previous sections can be used for the "required" relations (that capture the honest party strategy in the semi-malicious protocol π) by using a series of AND and OR proofs. We note that this suffices for our purposes since we can first transform these Σ-protocols into NIZKs in the RO model and then uses the resultant NIZKs to compile the semi-malicious protocol π.

Modifications. We make several modifications to our protocol π from Section 5. The first modification is that we now require that the distribution φ used to select secret key positions $\mathbf{s}_d^k[i] \leftarrow \varphi$ is just the uniform distribution over $\{0,1\}$. We rely on the result of [GKPV10], which shows that the LWE assumption remains secure even when the secret $\mathbf{s}$ is chosen from such distribution (for appropriate choices of other parameters). Secondly, in addition to the bounds $B_\chi, B_{smdg}^{eval}, B_{smdg}^{enc}, B_{smdg}^{dec}$ we will also have corresponding "starred" values (e.g. B_χ^*) which are super-polynomially larger (e.g. $B_\chi/B_\chi^* = \mathsf{negl}(\kappa)$). We will also have two additional bounds $B_r = 1$ and B_r^*. Lastly, we will require that the moduli q_d are prime, which allows us to rely on Claim 2.3 showing that the public-keys $\mathbf{p}_d^k$ will *commit* the party P_k to a unique secret key $\mathbf{s}_d^k$.

Proving "Honest" Behavior. We now examine each of the three rounds in protocol π, one by one, and describe how a party can prove that it is following the protocol "honestly" with some input $\mathbf{x}_k$ and some randomness r_k. Since we use *gap* Σ-protocols, there will be a gap between the noise-levels (e.g. B_χ) that honest parties need to use for their (zero-knowledge) security, and the "starred" noise levels (e.g. B_χ^*) that dishonest parties are guaranteed to be using. We describe the proof needed in each round of the protocol.

Round 1. Recall that in this round, each party P_k sends the following values (for every ℓ, d, i, τ):

1. individual public keys $\mathbf{p}_d^k$,
2. symmetric-key encryptions $\left(\mathbf{a}_{d,i,\tau}^\ell, b_{d,i,\tau}^{\ell,k}\right)$ of 0 under the secret key $\mathbf{s}_d^k$, and
3. approximate encryptions $\left(\mathbf{a}_{d,i,\tau}^k, b_{d,i,\tau}^{k,k}\right)$ of $2^\tau \cdot \mathbf{s}_{d-1}^k[i]$ under the secret key $\mathbf{s}_d^k$.

Then, in order to prove "honest behavior" in round 1, each party P_k must prove all of the following:

Well-formedness of public keys: The public key $\mathbf{p}_d^k$ is "well-formed" i.e. there exists a secret key $\mathbf{s}_d^k$ and noise $2\mathbf{e}_d^k \in [-B, B]^m$ such that $\mathbf{p}_d^k = \mathbf{A}_d \cdot \mathbf{s}_d^k + 2\mathbf{e}_d^k$. This is equivalent to showing $(A_d, \mathbf{p}_d^k) \in L_{\mathsf{lwe}}^B$. We use a gap-proof with $B = 2B_\chi$ for ZK and $B = 2B_\chi^*$ for Soundness.

Encryptions of 0: The values $\left(\mathbf{a}_{d,i,\tau}^\ell, b_{d,i,\tau}^{\ell,k}\right)$ are symmetric-key encryption of 0 under the (unique) secret key $\mathbf{s}_d^k$ corresponding to the public key $\mathbf{p}_d^k$. This is just showing that

$$(\mathbf{a}_{d,i,\tau}^\ell, b_{d,i,\tau}^{\ell,k}, 0, \mathbf{A}_d, \mathbf{p}_d^k) \in L_{\mathsf{approx}}^B.$$

We use a gap-proof with $B = 2B_\chi$ for ZK and $B = 2B_\chi^*$ for Soundness.

Well-formedness of evaluation key: The values $\left(\mathbf{a}^k_{d,i,\tau}, b^{k,k}_{d,i,\tau}\right)$ are "approximate" encryptions of $2^\tau \cdot \mathbf{s}^k_{d-1}[i]$ under the (unique) secret key $\mathbf{s}^k_d$ corresponding to public key $\mathbf{p}^k_d$, where $\mathbf{s}^k_{d-1}[i]$ is the i^{th}-component in the (unique) secret key $\mathbf{s}^k_{d-1}$ corresponding to $\mathbf{p}^k_{d-1}$.

That is, $b^{k,k}_{d,i,\tau} = \langle \mathbf{a}^k_{d,i,\tau}, \mathbf{s}^k_d \rangle + 2^\tau \cdot \mathbf{s}^k_{d-1}[i] + 2e^{k,k}_{d,i,\tau}$ for some noise $e^{k,k}_{d,i,\tau}$, and $\mathbf{p}^k_d$ and $\mathbf{p}^k_{d-1}$ are of the form as described above. Proving this is equivalent to showing:

$$\bigvee_{b\in\{0,1\}} \left\{ \quad (\mathbf{A}_{d-1}, \mathbf{p}^k_{d-1}) \in L^{B,i,b}_{\mathsf{bit}} \quad \wedge \quad \left(\mathbf{a}^k_{d,i,\tau} \ , \ b^{k,k}_{d,i,\tau} \ , \ 2^\tau \cdot b \ , \ \mathbf{A}_d \ , \ \mathbf{p}^k_d\right) \in L^B_{\mathsf{approx}} \right\}$$

We use a gap-proof with $B = 2B_\chi$ for ZK and $B = 2B^*_\chi$ for Soundness.

Round 2. Recall that in this round, each party P_k sends the following values:

1. Ciphertext tuples $(\boldsymbol{\alpha}^{\ell,k}_{d,i,j,\tau}, \beta^{\ell,k}_{d,i,j,\tau})$ that are used to determine the "joint" evaluation key.
2. Encryptions $c_{k,i} = (\mathbf{v}_{k,i}, w_{k,i})$ of each input bit $\mathbf{x}_k[i]$ under the "joint" public key $\mathbf{p}^*_0$.

Then, in order to prove "honest behavior" in round 2, each party P_k must prove the following:

Well-formedness of ciphertext tuples: The tuple $\left(\boldsymbol{\alpha}^{\ell,k}_{d,i,j,\tau}, \beta^{\ell,k}_{d,i,j,\tau}\right)$ is computed as:

$$\left(\boldsymbol{\alpha}^{\ell,k}_{d,i,j,\tau}, \beta^{\ell,k}_{d,i,j,\tau}\right) = \mathbf{s}^k_{d-1}[j] \cdot \left(\mathbf{a}^\ell_{d,i,\tau}, \beta^\ell_{d,i,\tau}\right) + \left(\mathbf{v}^{\ell,k}_{d,i,j,\tau}, w^{\ell,k}_{d,i,j,\tau} + 2e\right),$$

where $(\mathbf{a}^\ell_{d,i,\tau}, \beta^\ell_{d,i,\tau})$ are public values determined after round 1, $\mathbf{s}^k_d[j]$ is the j^{th} component of the secret key corresponding to the public key $\mathbf{p}^k_d$, and $(\mathbf{v}^{\ell,k}_{d,i,j,\tau}, w^{\ell,k}_{d,i,j,\tau} + 2e)$ is a "smudged" public-key encryption of 0 under the (joint) public key $\mathbf{p}^*_d$. This is equivalent to proving:

$$\bigvee_{b\in\{0,1\}} \left\{ \begin{array}{l} (\mathbf{A}_{d-1}, \mathbf{p}^k_{d-1}) \in L^{B,j,b}_{\mathsf{bit}} \ \wedge \\ \left(\ \mathbf{A}_d \ , \ \mathbf{p}^*_d \ , \quad \left[(\boldsymbol{\alpha}^{\ell,k}_{d,i,j,\tau}, \beta^{\ell,k}_{d,i,j,\tau}) - b \cdot (\mathbf{v}^\ell_{d,i,\tau}, w^\ell_{d,i,\tau})\right] \ \right) \in L^{B_r,B_e}_{\mathsf{enc}} \end{array} \right\}$$

We use a gap proof with $B = 2B_\chi$, $B_r = 1$, $B_e = 2B^{eval}_{smdg}$ for ZK and $B = 2B^*_\chi$, $B_r = B^*_r$, $B_e = 2B^{eval^*}_{smdg}$ for soundness.

Valid encryption of input bits: Lastly for each input ciphertexts $c_{k,i} = (\mathbf{v}_{k,i}, w_{k,i})$ we prove that it is a valid encryption of a bit $\mu \in \{0, 1\}$. This is equivalent to showing that:

$$(\mathbf{A}_0, \mathbf{p}^*_0, \mathbf{v}_{k,i}, w_{k,i}) \in L^{B_r,B_e}_{\mathsf{enc}} \quad \vee \quad (\mathbf{A}_0, \mathbf{p}^*_0, \mathbf{v}_{k,i}, w_{k,i} - 1) \in L^{B_r,B_e}_{\mathsf{enc}}$$

We use a gap proof with $B_r = 1$, $B_e = 2B^{enc}_{smdg}$ for ZK and $B_r = B^*_r$, $B_e = 2B^{enc^*}_{smdg}$ for soundness.

Round 3. In this round, for each output ciphertext $c^*_j = (\mathbf{v}_j, w_j)$, each party P_k sends a "decryption share" $w^k_j = \mathbf{v}_j \cdot \mathbf{s}^k_D + 2e_k$ where e_k is some noise, and $\mathbf{s}^k_D$ is the secret key corresponding to the public key $\mathbf{p}^k_D$. This is equivalent to proving:

$$(\mathbf{v}_j, w^k_j, 0, \mathbf{A}_D, \mathbf{p}^k_D) \in L^{B_1,B_2}_{\mathsf{approx}}$$

We use a gap proof with $B_1 = 2B_\chi$, $B_2 = 2B^{dec}_{smdg}$ for ZK and $B_1 = 2B^*_\chi$,$B_2 = 2B^{dec^*}_{smdg}$ for soundness.

F.5 Mind the Gap

In the above section we showed how to "prove honest behavior" using simple relations (along with AND and OR proofs) while also providing simple (gap) Σ-protocols for these relations. We need to argue that the *gap* does not harm security. Unfortunately, this does not follow in a "black-box" way from the analysis of our general compiler from semi-malicious to fully malicious security (Section E). To argue this property, we need to go back to our basic semi-malicious protocol (Theorem 5.2) and analyze security in the case that honest parties use noise-bounds $B_\chi, B_r = 1, B^{eval}_{smdg}, B^{enc}_{smdg}, B^{dec}_{smdg}$ while malicious parties can use super-polynomially larger "starred" bounds $B^*_\chi, B^*_r, B^{eval^*}_{smdg}, B^{enc^*}_{smdg}, B^{dec^*}_{smdg}$. We notice that the proof goes through the same way as before as long as the requirements needed for correctness (see Theorem C.1) now also hold for the "starred" noise values and the smudging noise added by honest parties is larger than the starred noise used by malicious parties.

Parameters. For full generality we can show the correctness/security of our scheme for any choice of params satisfying the following conditions (recall $B_\varphi = 1$ since the secret key is just bits):

1. Firstly, the parameters need to satisfy the conditions of Theorem C.1 for correctness so that the moduli q_d are sufficiently *large* in relation to the "starred" noise bounds. In particular we need some value ρ such that $\{\ q_{d-1}/q_d \geq \rho\ \}_{d\in[D]}$ and

$$\begin{aligned} \rho &\geq 2^{\omega(\log(\kappa))} \cdot \max\left\{ B^{enc^*}_{smdg}\ ,\ N^2 B^{eval^*}_{smdg}\ ,\ N^2 n^2\ ,\ N^3 B^*_\chi\ ,\ N^3 m B_\chi \right\} \\ q_D &\geq \max\left\{\ 2\rho\cdot(Nn+2)\ ,\ 2(\rho + 2NB^{dec^*}_{smdg}+1)\ \right\} \end{aligned}$$

2. Secondly, we need the "smudging" noise added by honest parties to be sufficiently *large*. Specifically:

$$B^{dec}_{smdg} \geq 2^{\omega(\log(\kappa))}\cdot\rho \quad , \quad B^{enc}_{smdg}, B^{eval}_{smdg} \geq 2^{\omega(\log(\kappa))}\cdot B^*_\chi.$$

3. Thirdly, we need the parameters $\mathsf{params}_d = (n, m, q_d, \varphi, \chi)$ to be chosen so that the underlying encryption scheme E is semantically secure with pseudorandom ciphertexts when instantiated with params_d for all $d \in \{0,\ldots,D\}$. Furthermore, we need the public-key to uniquely determine a secret key as in Claim 2.3. In particular:

   - We set $m \geq (n+1)\log(q_0) + \omega(\log(\kappa))$.
   - All moduli q_d are odd *primes*.

Given params as above, security will follow if the $\mathrm{LWE}_{n,q_d,\varphi,\chi}$ assumption holds, where φ is uniformly random over $\{0,1\}$, for all $d \in \{0,\ldots,D\}$. By [GKPV10] this follows from standard LWE where φ is uniformly random over $\mathbb{Z}_{q_d}$ with slightly dimension n.

G Fairness

In Section 4 we provided a multiparty computation protocol that is secure-with-abort, meaning that the adversary may learn the output alone without the honest parties. In this section, we show how to modify the construction in order to obtain full security (with fairness). In order to achieve

this, we cannot tolerate any number of corrupted parties, and we assume an honest majority (full security without an honest majority is impossible to achieve [Cle86]). We also assume authenticated broadcast channel, where the adversary cannot modify (or omit) messages sent by honest parties.

In order to achieve full security, the parties should proceed with the execution of the protocol even when some of the parties abort. In particular, the parties should be able to decrypt the common ciphertext even when some of the parties do not participate. Therefore, we also need to modify the construction to be a *threshold* scheme. The original scheme can be seen as an N-out-of-N scheme, meaning that all the parties should participate in the decryption. On the other hand, the modified scheme can be seen as a t-out-of-N scheme, meaning that any subset of t parties is able to decrypt a common ciphertext, but any subset of less than t parties cannot gain any information about the underlying message of a common ciphertext.

Building block – secret sharing scheme. An important building block for our solution is a secret sharing scheme [Sha79]. Loosely speaking, a t-out-of-N secret sharing scheme enables a dealer to distribute some secret s, such that each party P_i, $i \in [N]$ receives one share s_i. Every subset of t parties should be able to reconstruct s from their shares, while the secrecy requirement is that any subset of less than t parties cannot learn any information about s from the shares they hold.

The protocol. The protocol should enable the parties to finish successfully the execution even when some minority of the parties abort, while we are still interested in having minimal number of rounds. In general, in order to continue the execution after some party has aborted, the parties should be able to simulate this party's action. This is usually done by secret sharing: at the first round each party distributes shares of its random tape and its input. Once a party has aborted, the parties are able to reconstruct its input and randomness, and to (deterministically) compute its actions until the end of the execution. Observe that making this information public does not give the adversary any new information, since this public information is in fact part of the adversary's secret information.

We follow almost the same technique as above, and we now show how to instantiate this general idea to our protocol, and show the necessary modifications for the Keygen and Dec protocols, as well as to the general MPC protocol.

We begin with the Keygen protocol. In this protocol each party does not have any private input. Moreover, an important property of the protocol (and protocols that are secure with respect to semi-malicious adversary), is that the protocol is secure even if the adversary choose worst case random coins, and the randomness is not chosen in advanced. Therefore, we can avoid the sharing of the random tapes in the beginning of the protocol. Instead, a party needs to share its internal-state, which is, in fact, the individual private keys – $\mathbf{s}_0^k, \ldots, \mathbf{s}_D^k$. In case a party aborts, the parties reconstruct its internal state, and run the next message function with some common arbitrary coins. Therefore, any abortion is equivalent to an execution of the protocol with some different random tape. We describe formally the differences.

The Keygen protocol.

- **Round 1:** In addition to all messages that are broadcasted, each party P_k distributes $\{\mathbf{s}_d^k\}_d$ using the regular secret sharing scheme, with threshold $N/2+1$. Let $ss_d^{\ell,k}$ be the P_ℓ's share of $\mathbf{s}_d^k$, for every $d \in \{0, \ldots, D\}$, $k, \ell \in [N]$.

- **End of Round 1:** In case some party aborts before sending its message in round 1 – the party is just ignored, and the parties continue running the protocol like this party has not participated at all.
- **Round 2:** Exactly at the original protocol.
- **End of round 2:** In case some party P_ℓ aborts before sending its message in round 2, the parties proceed to round 3. Otherwise, they can terminate the execution.
- **Round 3 – in case of abort only:** For each party P_ℓ that did not broadcast its message in round 2, each party P_k broadcasts its shares $\{ss_d^{k,\ell}\}_d$. Giving all the shares that were broadcasted, each party can reconstruct the secrets $\mathbf{s}_0^\ell, \ldots, \mathbf{s}_D^\ell$, and compute the messages P_ℓ should have sent in round 2. This is done by just invoking the next message function on the internal state $\mathbf{s}_0^\ell, \ldots, \mathbf{s}_D^\ell$, and the messages that were broadcasted in round 1.
- Output: each party defines as decryption share: $(\mathbf{s}_D^k, ss_D^{1,k}, \ldots, ss_D^{N,k})$, instead of just $\mathbf{s}_D^k$.

Note that in case the adversary does not abort, the protocol takes 2 rounds. However, in the worst case, the execution takes one additional round.

The decryption protocol Dec. We show the necessary changes:

- **Input:** Each party P_k holds a share of the secret key $\mathbf{s}_D^k$ and the sub-shares $(ss_D^{1,k}, \ldots, ss_D^{N,k})$. (recall that the actual input in the original protocol is $\mathbf{s}_D^k$ only).
- **Round 1:** Same as the original protocol.
- **Round 2 – in case of abort only:** For each party P_ℓ that did not broadcast its message in round 1, each party P_k broadcasts $ss_D^{\ell,k}$. Giving all the broadcasted shares, reconstruct $\mathbf{s}_D^\ell$. Then, compute the message that P_ℓ should have sent using the private input $\mathbf{s}_D^\ell$ and arbitrary randomness 0.

Note that in the optimistic case, where the adversary does not abort prematurely, the protocol takes one round of interaction only. In case where the adversary does abort, the interaction takes one additional round, where the parties simply reconstruct the adversary's data.

The general multiparty computation protocol. In addition, in round 2 each party is instructed to broadcast encryption of its input using the joint public-key. Once a party does not broadcast messages in round 2, the parties just take a default input 0 and arbitrary random coins, compute an encryption of 0 using this common coins, and take the result as the encryption of the input.

Giving a function $f : (\{0,1\}^*)^N \to \{0,1\}^*$, let π_f be the protocol after applying the necessary changes in Keygen, Dec and the general protocol. Since some parties can abort in Keygen, and some can abort in Dec, we have that in worst case the overall protocol takes 5 rounds of interaction, while in the optimistic case, it takes 3 rounds only. We have the following Theorem:

Theorem G.1. *Let $f : (\{0,1\}^*)^N \to \{0,1\}^*$ be any deterministic poly-time function and* params *satisfy the above restrictions. Then protocol π_f securely and fairly realizes $\mathcal{F}_f$ in the presence of a* semi-malicious *adversary (Definition A.2), corrupting any $t \leq N/2$ parties.*

Proof Sketch: Clearly, in case where the adversary does not abort the theorem hold, relying on the proof of Theorem 5.2. Next, consider the case where some of the corrupted parties abort. In such a case, the honest parties reconstruct the missing information, and the resulting of the execution is equivalent to an execution with some different randomness and inputs of some different semi-malicious adversary that does not abort. Since execution of adversaries that do not abort is indistinguishable, we get that the ensembles for our case are indistinguishable as well. ■