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

Gilad Asharov[1] **, Abhishek Jain[2], Adriana López-Alt[3], Eran Tromer[4] ***, Vinod Vaikuntanathan[5] †, and Daniel Wichs[6]

[1] Bar-Ilan University
[2] University of California Los Angeles (UCLA)
[3] New York University (NYU)
[4] Tel Aviv University
[5] University of Toronto
[6] IBM Research, T.J. Watson

Abstract. Fully homomorphic encryption (FHE) enables secure computation over the encrypted data of a single party. We explore how to extend this to multiple parties, using *threshold* fully homomorphic encryption (TFHE). In such scheme, the parties jointly generate a common FHE public key along with a secret key that is shared among them; they can later cooperatively decrypt ciphertexts without learning anything but the plaintext. We show how to instantiate this approach efficiently, by extending the recent FHE schemes of Brakerski, Gentry and Vaikuntanathan (CRYPTO '11, FOCS '11, ITCS '12) based on the *(ring) learning with errors* assumption. Our main tool is to exploit the property that such schemes are additively homomorphic over their *keys*.

Using TFHE, we construct simple multiparty computation protocols secure against fully malicious attackers, tolerating any number of corruptions, and providing security in the universal composability framework. Our protocols have the following properties: ***Low interaction***: 3 rounds of interaction given a common random string, or 2 rounds with a public-key infrastructure. ***Low communication:*** independent of the function being computed (proportional to just input and output sizes). ***Cloud-assisted computation:*** the bulk of the computation can be efficiently outsourced to an external entity (e.g. a cloud service) so that the computation of all other parties is independent of the complexity of the evaluated function.

* This paper is a merger of two independent but largely overlapping works [3, 25]. The respective full versions are posted on ePrint.

** Supported by the European Research Council as part of the ERC project LAST.

*** This work was partially supported by the Check Point Institute for Information Security and by the Israeli Centers of Research Excellence (I-CORE) program (center No. 4/11).

† This work was partially supported by an NSERC Discovery Grant and by DARPA under Agreement number FA8750-11-2-0225. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

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 [33, 34] who gave a protocol for the case of *two semi-honest* that follow the protocol specification honestly but wish to learn as much information as possible, and Goldreich, Micali and Wigderson [19] 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. 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 [34, 19].

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. [31], the first implementation is due to the recent breakthrough of Gentry [17], and has now been followed with much exciting activity, most recently with quite simple and efficient schemes [10, 9, 8]. Using FHE, we immediately get an alternative approach to MPC in the case of two semi-honest 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 [34]).[7]

MPC via Threshold FHE. Since FHE solves the secure computation problem for two semi-honest parties, it is natural to ask whether we can extend the above template to the general case of *many fully malicious* parties. Indeed, there is a simple positive answer to this question (as pointed out in e.g. [17]) 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. [19]). 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

[7] Indeed, Yao's garbled circuits can be thought of as instantiating an FHE with long ciphertexts (see e.g. [18]).

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.

1.1 Our Results

In this work, we present an efficient threshold FHE (TFHE) scheme under the learning with errors (LWE) assumption, based on the FHE constructions of Brakerski, Gentry and Vaikuntanathan [9, 8]. Our starting observation is that basic LWE-based encryption ([30]) is *key homomorphic*, where summing up several public/secret key pairs (pk_i, sk_i) results in a new valid public/secret key pair $(pk^*, sk^*) = \sum_i (pk_i, sk_i)$.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 the 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.

Main Challenges of TFHE. The first challenge in instantiating the above idea 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. 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.

Using TFHE for MPC. Our basic TFHE protocol allows us to achieve MPC in the semi-honest 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 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 [32] 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). 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 for the necessary relations, which we can then compile into efficient UC NIZKs in the *random-oracle (RO)* model. Therefore, we can get a reasonably *efficient* and very *simple* 3-round protocol for general MPC in the RO model.

Cloud-Assisted MPC. We notice that our protocol can also be easily adapted to the setting of "cloud-assisted computation", where an (untrusted) external entity (e.g. "the cloud") is tasked with performing the homomorphic evaluation over the publicly broadcast ciphertexts. This results in a protocol where the computation of all other parties is small and independent of the size of the evaluated function! This approach only incurs one additional round in which the server broadcasts the ciphertext. To get security against a fully malicious server, we also require the existence of *succinct non-interactive argument systems*.

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. This gives the first *two*-round MPC construction in the PKI setting. We note that the PKI can be *reused* for many MPC executions of arbitrary functions and arbitrary inputs.

We summarize the above discussion with the following informal theorem:

Main Theorem. (informal) *Under the LWE assumption and the existence of UC NIZKs, for any function f there exists a protocol realizing f that is UC-secure in the presence of a (static) malicious adversary corrupting any number of parties. The protocol consists of **3 rounds** of broadcast in the CRS model, or **2 rounds** in a PKI model. Under an additional "circular security" assumption, its **communication** complexity is independent of the size of the evaluated circuit. In the "cloud-assisted setting" the total **computation** of each party (other than the cloud) is* independent *of the complexity of f.*

1.2 Related Work

In the context of general MPC, starting from the original proposal of Yao [34], there has been a rich line of work studying the round-complexity of secure multi-party computation protocols. In the semi-honest case, Beaver, Micali and Rogaway [4] gave the first constant-round protocol, which is asymptotically optimal.

An alternative approach using randomized polynomials was also given by [22, 2]. Although the concrete constants were not explicitly stated, they seem to require at least 4 rounds.In the fully malicious case there is a lower bound of 5 rounds in the plain model (dishonest majority) [24], but it does not seem to extend to the CRS model or other setup models. In the CRS model, we can generically compile semi-honest secure protocols into the fully malicious model using coin-tossing and (UC) NIZKs [32], at the cost of adding *two extra rounds* for the coin-toss. Therefore, the best prior works seem to require at least 6 rounds in the CRS model, although the exact constants were never carefully analyzed.

Recently, Choi et al [11] obtained a UC secure protocol in a *"pre-processing"* model with a 2-round online stage. However, the pre-processing requires "expensive" computation of garbled circuits and can later be only used once for a single online computation (it is *not* reusable). In contrast, our results give a 2-round UC-protocol in the PKI model, which we can think of as "pre-processing" that is only performed once and may be reused for arbitrarily many computations.

The works of [12, 15, 6, 13] use *additively* and *somewhat* homomorphic encryption to get some of the most practically efficient MPC implementations. However, since the schemes are not fully homomorphic, the round and communication complexity in these works is large and linear in the depth of the circuit.

The work of Bendlin and Damgård [5] builds a threshold version of [30] encryption based on LWE and ZK protocols for plaintext knowledge. Indeed, the main ideas behind our *decryption* protocol, such as the idea of using extra noise for "smudging", come from that work. We seem to avoid some of the main difficulties of [5] 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.

In a concurrent and independent work, Myers, Sergi and shelat [28] instantiate a threshold FHE scheme 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 [28], 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.

The idea of using a cloud to alleviate the computational efforts of parties was recently explored in the work on "server-aided MPC" by Kamara, Mohassel and Raykova [23]. Their protocols, however, require some of the parties to do a large amount of computation, essentially proportional to the size of the function f being computed. Halevi, Lindell and Pinkas [21] recently considered the model of "secure computation on the web" which gets rid of all interaction between the actual parties, and instead only allows each party to "log in" once to interact with the server. Unfortunately, this necessitates a *weaker* notion of security which is only meaningful for a small class of functions. In contrast, we focus here on *standard* MPC security for arbitrary functions, at the cost of ad-

ditional interaction. In particular, we achieve full security in the model where the computation occurs in 2 stages (optimal) and each party "logs in" once per stage to post a message to the server. As an additional benefit, 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, unlike [21] where the parties must "log in" *sequentially*.

1.3 Organization

In the proceedings version of this work, we follow the exposition of [3], and all omitted proofs can be found there. See [25] for an alternative exposition, using somewhat different abstractions and a variant of the scheme presented here under the *ring LWE* assumption.

In Section 3 we start with a basic LWE-based encryption scheme, highlight its homomorphic properties, and describe how to use it to get the FHE schemes of [9, 8]. In Section 4 we then describe our threshold FHE scheme, and in Section 5 we use it to build an MPC protocol. We then discuss several variants of this protocol in Section 6.

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 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 1 (Smudging) *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 $B_1/B_2 = \mathsf{negl}(\kappa)$.*

Learning With Errors. The *decisional learning with errors (*LWE*)* problem, introduced by Regev [30], is defined as follows.

Definition 2 (LWE [30]) *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 two distributions on tuples $(\mathbf{a}_i, b_i)$, given polynomially many samples:* ***Distribution I.*** *Each $(\mathbf{a}_i, b_i) \stackrel{\$}{\leftarrow} \mathbb{Z}_q^{n+1}$ is chosen independently, uniformly at random.* ***Distribution II.*** *Choose $\mathbf{s} \leftarrow \varphi^n$. Each sample $(\mathbf{a}_i, b_i)$ is chosen as: $\mathbf{a}_i \stackrel{\$}{\leftarrow} \mathbb{Z}_q^n$, $e_i \leftarrow \chi$, $b_i := \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$.*

The works of [30, 29] 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 [1] shows that, when q is a prime power, then $\text{LWE}_{n,q,\chi,\chi}$ is as hard as $\text{LWE}_{n,q,U(\mathbb{Z}_q),\chi}$. Therefore, we can assume that the secret $\mathbf{s}$ of the LWE problem also comes from a "small" Gaussian distribution. It is also easy to see that, if q is odd, then $\text{LWE}_{n,q,\varphi,(2\chi)}$ is as hard as $\text{LWE}_{n,q,\varphi,\chi}$, where the distribution 2χ samples $e \leftarrow \chi$ and outputs $2e$.

3 Homomorphic Encryption from LWE

In this section, we give a brief description of the FHE schemes of [9, 8].

Basic LWE-based Encryption. We start by describing a basic symmetric/public encryption scheme E, which is a variant of [30] encryption scheme based on the LWE problem. This scheme serves as a building block for the more complex FHE schemes of [9, 8] and of our threshold FHE scheme.

- $\mathbf{params = (1^\kappa, q, m, n, \varphi, \chi)}$ **:** 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 $\mathbb{Z}_q$.
- **E.SymKeygen(params):** Choose a secret key $\mathbf{s} \leftarrow \varphi^n$.
- **E.PubKeygen(s):** 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 $\mathbf{s}$.
- $\mathbf{E.SymEnc_s(\mu)}$**:** 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 $c = (\mathbf{a}, b)$.
- $\mathbf{E.PubEnc_{pk}(\mu)}$**:** To encrypt a message $\mu \in \{0, 1\}$ under $pk = (A, \mathbf{p})$, choose $\mathbf{r} \leftarrow \{0, 1\}^m$ and set $\mathbf{a} \stackrel{\text{def}}{=} \mathbf{r}^T \cdot A$, $\quad b \stackrel{\text{def}}{=} \langle \mathbf{r}, \mathbf{p} \rangle + \mu$. Output $c = (\mathbf{a}, b)$.
- $\mathbf{E.Dec_s(c)}$ **– (decryption):** Parse $c = (\mathbf{a}, b)$, output $[b - \langle \mathbf{a}, \mathbf{s} \rangle]_q \bmod 2$.

Under appropriate parameters and LWE assumption, the above scheme is semantically secure *with pseudorandom 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 ([30]) *Assuming n, q, $m \geq (n+1)\log(q) + \omega(\log(\kappa))$ are integers with q odd, and that the* $\text{LWE}_{n,q,\varphi,\chi}$ *assumption holds, the above* public key encryption scheme $(\mathsf{E.PubKeygen}, \mathsf{E.PubEnc}, \mathsf{E.Dec})$ *is semantically secure with* pseudorandom ciphertexts.

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 $\mathsf{E.PubKeygen}(\mathbf{s}; A)$, $\mathsf{E.PubKeygen}(\mathbf{s}; A; \mathbf{e})$ to denote the execution of the key generation algorithm with fixed coefficients A and (respectively) with fixed A, e. We use $\mathsf{E.SymEnc}_{\mathbf{s}}(\mu; \mathbf{a})$, $\mathsf{E.SymEnc}_{\mathbf{s}}(\mu; \mathbf{a}; e)$ analogously.

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 notice it also satisfies several useful *key-homomorphic* properties, which make it particularly easy to convert into a threshold scheme. In particular, let $\mathbf{s}_1, \mathbf{s}_2$ be two secrets keys, $\mathbf{a}$ be some coefficient vector $(\mathbf{a}, b_1) = \mathsf{E.SymEnc}_{\mathbf{s}_1}(\mu_1; \mathbf{a})$, $(\mathbf{a}, b_2) = \mathsf{E.SymEnc}_{\mathbf{s}_2}(\mu_2; \mathbf{a})$ be two ciphertexts encrypting the bits μ_1, μ_2 under the keys $\mathbf{s}_1, \mathbf{s}_2$ respectively but using the same randomness $\mathbf{a}$. Then we can write $b_1 = \langle \mathbf{a}, \mathbf{s}_1 \rangle + 2 \cdot e_1 + \mu_1 \quad , \quad b_2 = \langle \mathbf{a}, \mathbf{s}_2 \rangle + 2 \cdot e_2 + \mu_2$ and

$$b^* := b_1 + b_2 = \langle \mathbf{a}, (\mathbf{s}_1 + \mathbf{s}_2) \rangle + 2(e_1 + e_2) + (\mu_1 + \mu_2).$$

So $(\mathbf{a}, b^*) = \mathsf{E.SymEnc}_{\mathbf{s}_1+\mathbf{s}_2}(\mu_1 + \mu_2; \mathbf{a})$ is an encryption of $\mu_1 + \mu_2$ under the sum of the keys $(\mathbf{s}_1 + \mathbf{s}_2)$ with a noise level which is just the sum of the noises. Also, if we keep the matrix A fixed, then the sum of two key pairs gives a new valid key pair. That is, if $\mathbf{p}_1 = A\mathbf{s}_1 + 2\mathbf{e}_1 \quad , \quad \mathbf{p}_2 = A\mathbf{s}_2 + 2\mathbf{e}_2$ are public key with corresponding secret keys $\mathbf{s}_1, \mathbf{s}_2$, then

$$\mathbf{p}^* := \mathbf{p}_1 + \mathbf{p}_2 = A(\mathbf{s}_2 + \mathbf{s}_2) + 2(\mathbf{e}_1 + \mathbf{e}_2)$$

is a public key for the corresponding secret key $\mathbf{s}^* = \mathbf{s}_1 + \mathbf{s}_2$.

Security of Joint Keys. We show a useful security property of combining public keys. Assume that a public key $\mathbf{p} = A\mathbf{s} + 2e$ 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 $\mathbf{p}^* = \mathbf{p} + \mathbf{p}'$ from uniformly random ones.[8] Note that the combined key $\mathbf{p}^*$ may not be at all distributed like a correct public key, and the attacker has a large degree of control over it. 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 following experiment $\mathsf{JoinKeys}_{\mathcal{A}}(\mathsf{params}, B_1, B_2)$:
(-) Challenger chooses $\mathbf{s} \leftarrow \mathsf{E.SymKeygen}(\mathsf{params})$, $(A, \mathbf{p}) \leftarrow \mathsf{E.PubKeygen}(\mathbf{s})$.
(-) $\mathcal{A}$ gets $(A, \mathbf{p})$ and 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 $\mu \in \{0, 1\}$.
(-) 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 $b^* = b + 2e^*$.
(-) $\mathcal{A}$ gets $(\mathbf{a}^*, b^*)$ and outputs a bit $\tilde{\beta}$.
The output of the experiment is 1 if $\tilde{\beta} = \beta$, and 0 otherwise.

[8] A similar idea was used in [16] in the context of threshold ElGamal encryption.

Lemma 4 *Let q, m, n, φ, χ be set as in Theorem 3 and assume that* $\mathrm{LWE}_{n,q,\varphi,\chi}$ *assumption holds. 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 $\mathcal{A}$*:* $|\Pr[\mathsf{JoinKeys}_{\mathcal{A}}(\mathsf{params}, B_1, B_2) = 1] - \frac{1}{2}| = \mathsf{negl}(\kappa)$.

3.1 Fully Homomorphic Encryption from LWE

In this section we present the construction of [9, 8]. 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.

- $\mathbf{FHE.Keygen}(1^\kappa) \rightarrow (\boldsymbol{pk}, \boldsymbol{evk}, \boldsymbol{sk})$: Outputs a public encryption key pk, a public evaluation key evk and a secret decryption key sk.
- $\mathbf{FHE.Enc}_{\boldsymbol{pk}}(\boldsymbol{\mu})$,$\mathbf{FHE.Dec}_{\boldsymbol{sk}}(\boldsymbol{c})$: Have the usual syntax of public-key encryption/decryption.
- $\mathbf{FHE.Eval}_{\boldsymbol{evk}}(\boldsymbol{f}, \boldsymbol{c_1}, \ldots, \boldsymbol{c_\ell}) = \boldsymbol{c_f}$: The *homomorphic evaluation algorithm* is a *deterministic* poly-time algorithm that takes the evaluation key evk, a boolean circuit $f : \{0,1\}^\ell \rightarrow \{0,1\}$, 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. We say that it is *fully homomorphic* if for any boolean circuit $f : \{0,1\}^\ell \rightarrow \{0,1\}$ and respective inputs $\mu_1, \ldots, \mu_\ell \in \{0,1\}$, keys $(pk, evk, sk) \leftarrow \mathsf{FHE.Keygen}(1^\kappa)$ and ciphertexts $c_i \leftarrow \mathsf{FHE.Enc}_{pk}(\mu_i)$ it holds that: $\mathsf{FHE.Dec}\left(\mathsf{FHE.Eval}_{evk}\left(f, c_1, \ldots, c_\ell\right)\right) = f(\mu_1, \ldots, \mu_\ell)$. 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 [9, 8]. 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 $\phi_c(\mathbf{x}) \stackrel{\text{def}}{=} b - \langle \mathbf{a}, \mathbf{x} \rangle$: an n-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 (recall, we define $\mathbf{s}[0] \stackrel{\text{def}}{=} 1$):

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

See [9] 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 $D+1$ 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 [8] gives a beautifully simple 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. 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 [17], 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 an individual private share sk_k of the secret key. 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 in the context of our general MPC protocol 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 [8] 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:
1. $\mathsf{params} = \left(\{\mathsf{params}_d\}_{0\le d\le D}\ ,\ B_\varphi, B_\chi, B^{eval}_{smdg}, B^{enc}_{smdg}, B^{dec}_{smdg}\right)$, where

- $\mathsf{params}_d = (1^\kappa, q_d, m, n, \varphi, \chi)$ 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.

2. 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).

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 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 $\left\{\mathbf{p}_d^k\right\}_d$, $\left\{b_{d,i,\tau}^{\ell,k}\right\}_{\ell,d,i,\tau}$.

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: $\mathbf{p}_d^* := \sum_{\ell=1}^N \mathbf{p}_d^\ell$. Let $pk := (A_0, \mathbf{p}_0^*)$ be the common public encryption key of the TFHE scheme.

   Notice that, if all parties act honestly then $(A_d, \mathbf{p}_d^*) = \mathsf{E.PubKeygen}(\mathbf{s}_d^*; A_d; \mathbf{e}_d^*)$. where $\mathbf{s}_d^* := \sum_{\ell=1}^N \mathbf{s}_d^\ell$, $\mathbf{e}_d^* := \sum_{\ell=1}^N \mathbf{e}_d^\ell$. 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 $\beta_{d,i,\tau}^\ell := \sum_{k=1}^N b_{d,i,\tau}^{\ell,k}$.

   Notice that, if all parties follow the protocol then:

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

   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* $\mathbf{s}_{d-1}^\ell$ 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 $(\mathbf{v}_{d,i,j,\tau}^{\ell,k}, w_{d,i,j,\tau}^{\ell,k}) \leftarrow \mathsf{E.PubEnc}_{p_d^*}(0)$ and $e \xleftarrow{\$} [-B_{smdg}^{eval}, B_{smdg}^{eval}]$. Set:

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

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

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:

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

   Note that, if all parties follow the protocol, then

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

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

Outputs:

1. *Public Evaluation key:* Output $evk = \{\psi_{d,i,j,\tau}\}_{d,i,j,\tau}$.
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 $\mathbf{s}_D^k$.

$\mathsf{TFHE.Enc}_{pk}(\mu)$: 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" $e \xleftarrow{\$} [-B_{smdg}^{enc}, B_{smdg}^{enc}]$ 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 [8].

The Decryption Protocol: **$\mathsf{TFHE.Dec}(c)$.** This is a one-round protocol between N parties. Initially all parties hold a common ciphertext $c = (\mathbf{v}, w, D)$ with associated "level" D. 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 $w^k = \langle \mathbf{v}, \mathbf{s}_D^k \rangle + 2 \cdot e_k$, where $e_k \xleftarrow{\$} [-B_{smdg}^{dec}, B_{smdg}^{dec}]$.
- Given $w^1, \ldots, w^N$, compute the output bit: $\mu = [w - \sum_{i=1}^N w^i]_{q_D} \mod 2$.

5 Secure MPC via TFHE

We now present a protocol for general MPC, using the threshold fully homomorphic scheme TFHE from the previous section. Let $f : (\{0,1\}^{\ell_{in}})^N \to \{0,1\}^{\ell_{out}}$ be a deterministic function computed by a circuit of multiplicative depth D. Let $(\mathsf{TFHE.Keygen}, \mathsf{TFHE.Enc}, \mathsf{TFHE.Eval}, \mathsf{TFHE.Dec})$ be our TFHE scheme from the previous section, initiated for D levels, and with parameters setup. Our basic MPC protocol π_f for evaluating the function f proceeds as follows.

Initialization: Each party P_k has input $\mathbf{x}_k \in \{0,1\}^{\ell_{in}}$. The parties share the common parameters setup for our D-level TFHE scheme.

Round I. The parties execute the *first* round of the $\mathsf{TFHE.Keygen}$ 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 $\mathsf{TFHE.Keygen}$ 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 locally computes the 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 $\mathsf{TFHE.Dec}$ 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.

Security for Semi-Malicious Attackers. We show that the above protocol is secure against a semi-honest attacker corrupting any number of parties. Actually, we show security against a stronger class of attackers which we call *semi-malicious*. A semi-malicious attacker follows the honest protocol specification but with some adversarially chosen random coins (of which it has knowledge). It can choose its malicious random coins adaptively in each round after seeing the protocol messages of all honest parties during that round. We state our main theorem without concrete parameters. We defer the proof to the full version, where we also discuss the settings of the parameters for our protocol and the corresponding LWE assumption required for security.

Theorem 5 *Let f be any deterministic poly-time function with N inputs and single output (same output for all parties). Then there is a setting of parameters* params *such that, under the corresponding LWE assumption, the protocol π_f securely UC-realizes f in the presence of a static* semi-malicious *adversary corrupting any $t \leq N$ parties.*

Proof Intuition. We now give a high-level description of how the proof of security works, and relegate the proof to the full version. 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 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 simulated 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.

Security for Fully Malicious Attackers. Our basic MPC protocol is only secure in the semi-malicious setting. In the full version, we give a general round-preserving compiler from semi-malicious to fully malicious security using UC NIZKs [32] 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. Combining this with Theorem 5, we get a 3 round MPC protocol in the CRS model for a fully malicious attacker corrupting any number of parties.

In the full version (see [3]), we also address the question of instantiating such NIZKs *efficiently*. We first present simple, efficient, and statistical Σ-protocols for basic LWE-languages. 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 behaving "honestly". We can then compile them into UC-NIZKs and obtain general 3-round MPC protocols, in the random oracle model.

6 Variants and Optimizations

We consider several variants and optimizations of our basic MPC protocol.

Two Round MPC under a 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).

Cloud-Assisted Computation. Our protocol can be made extremely efficient by outsourcing large public computations. In particular, the only intensive computation in our protocol, that depends on the circuit size of the evaluated function,

is the homomorphic evaluation at the end of round II. In our basic description of the protocol, we assumed that each party performs this computation *individually*, but we notice that this computation is the same for all parties and does not require any secret inputs. Therefore, it is possible to designate one special party P^* (or even an external entity e.g. a powerful server, or the "cloud") that does this computation on everyone's behalf and broadcasts the resulting output ciphertexts to everyone else. Moreover, if P^* is one of the parties, it does not need to broadcast its input ciphertexts to everyone else in round II, since 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. Broadcasting the output ciphertexts requires an extra round, raising the round complexity of this variant to 4 rounds in the CRS model, 3 rounds in PKI model.

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 or under strong assumptions [27, 7, 20, 14].

Ring LWE. In the full version (see [25]), we show a variant of the protocol using *ring* LWE [26]. This variant provides significant practical efficiency savings over just using standard LWE and the resulting scheme may be even conceptually simpler than using standard LWE.

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 [17]. To apply the 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 depending on the decryption circuit and maintain the ability to homomorphically evaluate arbitrarily large function f. 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.

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.

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.

References

1. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: CRYPTO. pp. 595–618 (2009)
2. Applebaum, B., Ishai, Y., Kushilevitz, E.: Computationally private randomizing polynomials and their applications. In: IEEE Conference on Computational Complexity. pp. 260–274 (2005)
3. Asharov, G., Jain, A., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold fhe. Cryptology ePrint Archive, Report 2011/613 (2011), http://eprint.iacr.org/
4. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: STOC. pp. 503–513 (1990)
5. Bendlin, R., Damgård, I.: Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In: TCC (2010)
6. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: EUROCRYPT. pp. 169–188 (2011)
7. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: ITCS (2012)
8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. In: ITCS (2012)
9. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. In: FOCS (2011)
10. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: CRYPTO. pp. 505–524 (2011)
11. Choi, S.G., Elbaz, A., Malkin, T., Yung, M.: Secure multi-party computation minimizing online rounds. In: ASIACRYPT (2009)
12. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: EUROCRYPT (2001)
13. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535 (2011), http://eprint.iacr.org/
14. Damgård, I., Faust, S., Hazay, C.: Secure two-party computation with low communication. In: TCC (2012)
15. Damgård, I., Nielsen, J.B.: Universally composable efficient multiparty computation from threshold homomorphic encryption. In: CRYPTO. pp. 247–264 (2003)
16. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure applications of pedersen's distributed key generation protocol. In: CT-RSA. pp. 373–390 (2003)

17. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC. pp. 169–178 (2009)
18. Gentry, C., Halevi, S., Vaikuntanathan, V.: i-hop homomorphic encryption and rerandomizable yao circuits. In: CRYPTO. pp. 155–172 (2010)
19. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC. pp. 218–229 (1987)
20. Goldwasser, S., Lin, H., Rubinstein, A.: Delegation of computation without rejection problem from designated verifier cs-proofs. Cryptology ePrint Archive, Report 2011/456 (2011), http://eprint.iacr.org/
21. Halevi, S., Lindell, Y., Pinkas, B.: Secure computation on the web: Computing without simultaneous interaction. In: CRYPTO. pp. 132–150 (2011)
22. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: FOCS. pp. 294–304 (2000)
23. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272 (2011), http://eprint.iacr.org/
24. Katz, J., Ostrovsky, R.: Round-optimal secure two-party computation. In: CRYPTO. pp. 335–354 (2004)
25. López-Alt, A., Tromer, E., Vaikuntanathan, V.: Cloud-assisted multiparty computation from fully homomorphic encryption. Cryptology ePrint Archive, Report 2011/663 (2011), http://eprint.iacr.org/
26. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: EUROCRYPT. pp. 1–23 (2010)
27. Micali, S.: Computationally sound proofs. SIAM J. Comput. 30(4), 1253–1298 (2000)
28. Myers, S., Sergi, M., Shelat, A.: Threshold fully homomorphic encryption and secure computation. In: eprint 2011/454 (2011)
29. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: STOC. pp. 333–342 (2009)
30. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC. pp. 84–93 (2005)
31. Rivest, R., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. In: Foundations on Secure Computation, Academia Press. pp. 169–179 (1978)
32. Santis, A.D., Crescenzo, G.D., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: CRYPTO. pp. 566–598 (2001)
33. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: FOCS. pp. 160–164 (1982)
34. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: FOCS. pp. 162–167 (1986)