Constant-Round MPC with Fairness and Guarantee of Output Delivery

S. Dov Gordon[*] Feng-Hao Liu[†] Elaine Shi[‡]

April 22, 2015

Abstract

We study the round complexity of multiparty computation with fairness and guaranteed output delivery, assuming existence of an honest majority. We demonstrate a new lower bound and a matching upper bound. Our lower bound rules out any two-round fair protocols in the standalone model, even when the parties are given access to a common reference string (CRS). The lower bound follows by a reduction to the impossibility result of virtual black box obfuscation of arbitrary circuits.

Then we demonstrate a three-round protocol with guarantee of output delivery, which in general is harder than achieving fairness (since the latter allows the adversary to force a fair abort). We develop a new construction of a threshold fully homomorphic encryption scheme, with a new property that we call "flexible" ciphertexts. Roughly, our threshold encryption scheme allows parties to adapt flexible ciphertexts to the public keys of the non-aborting parties, which provides a way of handling aborts without adding any communication.

1 Introduction

Secure multi-party computation (MPC) allows mutually distrusting parties to securely compute a function on their inputs with several desired properties, including: **correctness** (honest parties should not receive a wrong output), and **privacy** (corrupted parties cannot learn anything beyond the prescribed output). In addition to these two basic properties, one might further require **fairness** (corrupted parties receive their output only if all honest parties receive output), or the stronger **guarantee of output delivery** (corrupted parties cannot prevent honest parties from receiving their output). Alternatively, a relaxed security notion is often used, called **security with abort** – it is possible that the attacker can prevent the honest parties from receiving output. All of these requirements can be formalized in an Ideal/Real paradigm [Can01, Gol04], which provides a nice way to analyze security.

In this work, we explore the *round complexity* required for achieving these various properties. For the setting of *security with abort*, we already understand the round-complexity fairly well – Asharov et al. [AJL$^+$12] constructed a 3-round protocol (in the common reference string model) under the learning with error (LWE) assumption; Garg et al. [GGHR14] constructed a 2-round protocol for general computation (in the CRS model) using indistinguishable obfuscation; it is well-known that one-round protocols are in general not possible.

However, for protocols with *fairness* and *guarantee of output delivery*, our understanding of round complexity is still incomplete. Regarding *feasibility*, everything is well understood: if there is no honest majority, Cleve proved [Cle86] that fair MPC for general computation is not possible. In the setting of an honest majority, we know that we can always achieve fairness [BOGW88], and, assuming a broadcast channel, we can always guarantee output delivery [CL14]. However, the optimal round complexity is still unknown. Asharov et al. [AJL$^+$12] show that their basic protocol can be extended to achieve security with guarantee of output delivery (and thus fairness) in 5 rounds, assuming there exists an honest majority. Garg et al. [GGHR14]

[*]ACS. Email: sgordon@appcomsci.com
[†]Dept. of Computer Science, University of Maryland. Email: fenghao@cs.umd.edu
[‡]Dept. of Computer Science, University of Maryland. Email: elaine@cs.umd.edu

claimed that their 2-round protocol also achieves fairness, however this is not true as we will demonstrate later. Thus, when $t < N/2$ parties can be corrupted, the best known round complexity is 5. We do not know of any non-trivial lower-bounds on round complexity in the honest majority setting.

1.1 Our Results

Our main two results are matching upper and lower bounds for three-round multiparty computation with guaranteed output delivery with security against a malicious minority of parties. More specifically:

- We show that 2-round, fair MPC for general functions is impossible, even if there is an honest majority. The impossibility result also rules out protocols in the CRS model. (Section 3.)
- There exists a 3-round MPC with guaranteed output delivery for general functions in the CRS model, secure against a minority of semi-honest fail-stop adversaries. The security relies on the learning with errors (LWE) assumption. (Section 4.)
- If parties have access to an authenticated broadcast channel[1], then the above 3-round protocol can be upgraded to one that is secure against malicious adversaries, without any additional rounds. (Section 5.)
- Additionally, we show that security of the two-round protocol by Garg et al. [GGHR14] can be based on witness encryptions for general NP statements, which is weaker than indistinguishable obfuscation for general circuits[2] as presented in their work. (Appendix D.) Together with an idea in Section 1.2, we can construct a three-round *fair* protocol (but not guarantee of output delivery) based on witness encryptions for general NP statements.

In summary, 2-round general fair MPC is not possible, and 3-round general MPC with guarantee of output delivery can be constructed under a falsifiable assumption. Guarantee of output delivery implies fairness (by definition), and thus 3-round fair MPC can also be constructed under the same falsifiable assumption.

All of our positive results are UC-secure [Can01]. Our protocols, along with those appearing in the prior work of Garg et al. [GGHR14] and Asharov et al. [AJL$^+$12], require a CRS.

1.2 Overview of our Techniques

Impossibility of Fairness in Two Rounds. We show that a two round, fair, polynomial-time protocol for general functions yields a construction of virtual black box (VBB) secure program obfuscation for P/Poly, in contradiction of the well-known impossibility result of Barak et al. [BGI$^+$01].

Consider a symmetric 3-ary functionality $f(x_1, x_2, x_3)$ that interpret x_1 as a circuit C, ignores x_3 and outputs $C(x_2)$. Suppose there exists a two-round fair protocol π that computes f with fairness, then we make the following observations. We assume the three parties are Alice, Bob, and Charlie.

- If the adversary (only) corrupts Alice and instructs her to abort in the second round, then after Bob and Charlie send their messages in the second round, the adversary can learn the output $C(x_2)$.
- By the property of fairness, Bob and Charlie must be able to learn the output $C(x_2)$, since the adversary in the above case has learned the output.
- It follows that Alice's second message is redundant. Whether she sends her second message or not, the other parties can compute the outcome.

Using the above observations, we can construct a program obfuscator for general circuits: we view Alice's first message as the obfuscation of C. To evaluate $C(x)$, we just simulate Bob and Charlie with Bob's input x. Since Alice's first message is independent of the other parties' inputs, we can rewind Bob and Charlie and compute $C(x)$ repeatedly on arbitrary values of x.

[1]An authenticated broadcast channel enables a party to send a message to all other parties, ensuring that each party knows both the identity of the sender and that all other parties have received the same message.

[2]Indistinguishable obfuscation for general circuits is a stronger assumption. We know that indistinguishable obfuscation for general circuits implies witness encryption for general NP statements, but the other way is unclear.

We note that Garg et al. state (without proof) that their two-round protocol achieves fairness [GGHR14], and one can see why this mistake might have been made. Their protocol works by collapsing some protocol with greater round complexity into a two-round protocol through the use of obfuscation, and they state that if the underlying protocol is fair, then the resulting two round protocol will also be fair. Speaking very roughly, in their construction, each party sends a commitment to their input and their randomness in round one, and in round two, they each send obfuscations of the next message functions from the underlying fair protocol. They then each finish the protocol locally, using the obfuscated programs to generate the correct protocol messages. At first glance, it would seem that this preserves fairness, because if a party aborts in round two, the other parties can simply generate next-messages as though he aborted, and, by the fairness of the underlying protocol, fairness should be preserved. In fact, this misses the following subtlety. If a party aborts in round two, he still receives all of the obfuscated programs, and can still compute the output of the function: this is equivalent to aborting in the very last round of the underlying fair protocol. On the other hand, because he never sent his obfuscated next-message programs, the other parties will be forced to treat him as though he aborted in round one of the underlying protocol, perhaps replacing his input with some default value. In particular, then, it could be that the malicious party learns $f(x_1, \ldots, x_N)$ while the other parties learn $f(\perp, x_2, \ldots, x_N)$.

Fairness in Three Rounds. The construction of Garg et al. can be modified slightly to get a three-round fair protocol, as we now outline. However, we note that there is no clear way to guarantee output delivery without increasing the round complexity; our main technical result is a new protocol for achieving guaranteed output delivery in three rounds.

To achieve fairness in three rounds, we can start with the protocol of Garg et al., but instead of sending obfuscations of the next message functions that compute the underlying secure computation, the parties will send obfuscations that compute an $N/2$-out-of-N secret sharing of the output. They then add one additional round to reconstruct the output. Now, if the adversary aborts in round two, even though he learns all of the next message functions, he still cannot recover the output (since there is an honest majority). If he aborts in round three, the honest parties already have enough shares to reconstruct the output on their own.

In general, we can compile any fair protocol into one that guarantees output delivery, assuming a broadcast channel [CL14], but we cannot necessarily preserve the round complexity. In the particular protocol just described, note that the obfuscated programs sent in round two have commitments to the parties' inputs embedded inside of them. If a party aborts in round two, the other parties would need to replace their obfuscations, embedding a commitment to some default input value in place of the aborting party's true input. But this will incur additional communication rounds.

Guarantee of Output Delivery. Before we describe our protocol, we first give an overview the approach by Asharov et al. [AJL$^+$12]. Asharov et al. proposed a new primitive called *Threshold Fully Homomorphic Encryption* (TFHE), which is essentially a distributed version of fully homomorphic encryption (FHE). For their TFHE, there is a joint public key pk^* whose secret key is shared among all parties, i.e. $\mathsf{sk}^* = \mathsf{sk}_1 + \mathsf{sk}_2 + \cdots + \mathsf{sk}_N$. (There is also an evaluation key, but we omit it for simplicity of exposition). The keys $(\mathsf{pk}^*, \mathsf{sk}^*)$ constitute an FHE key pair, so the encryption and evaluation algorithms can remain the same as those used in the original FHE scheme. To decrypt, parties need to run a *threshold decryption protocol*, since the secret key is shared among all parties.

Using the TFHE scheme, their basic three round protocol has the following structure: (1) in the first round parties establish a joint public key pk^*; (2) in the second round parties output an encryption of their inputs, i.e. $\mathsf{Enc}_{\mathsf{pk}^*}(x_i)$; (3) in the third round, parties perform the homomorphic operations (for computing f) to obtain an evaluated ciphertext C^*, and then run the threshold decryption protocol to decrypt C^*. Asharov et al. [AJL$^+$12] presented a simple idea to make the basic protocol fair (and to guarantee output delivery) in the first round, the parties also secret share their inputs and all random coins. If any party aborts in the second or third round, the honest majority would reconstruct his states and resume the protocol. (Note that if a party aborts in the first round, he is simply ignored). This approach will add two additional rounds for the worst case.

We note that their construction uses an N-out-of-N sharing of the secret key sk^*, so they require all parties in order to decrypt. This means, if any party aborts, the other parties need to reconstruct his view to resume. Thus, these two additional rounds seem inherent if we follow this approach. To get a 3-round protocol, we need a new approach. In particular we propose and construct a new variant of TFHE with more fine-grained features. Using it as a building block, we are able to get around the barriers mentioned above. We highlight our new ideas below.

Instead of establishing a “fixed” joint public key, our new TFHE uses $\mathsf{pk}_{[N]} = \{\mathsf{pk}_i\}_{i\in[N]}$ as the public keys, where pk_i is contributed by party P_i. Then with $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_N\}$, P_i can encrypt the input x_i and produce a **flexible** ciphertext C_i. We introduce a new algorithm $\mathsf{TransCT}(C; S)$ that transforms a flexible ciphertext C into C', where C' is with respect to the public keys $\mathsf{pk}_S = \{\mathsf{pk}_j : j \in S\}$. Intuitively, a *flexible* ciphertext is one that is not yet committed to a set of public keys, and a *transformed* one commits to some pk_S and can be homomorphically evaluated. Finally, our threshold decryption protocol works when there is an honest majority of parties (as opposed to the previous one which requires all parties).

Using the new TFHE, our protocol has the following structure: (1) in the first round, parties generate $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_N\}$; (2) in the second round, each party output a flexible ciphertext $C_i = \mathsf{Enc}(x_i)$; (3) let S be the parties that did not abort in the second round. Now each party transforms the ciphertexts to C'_i with respect to pk_S and performs the homomorphic evaluation for computing f. Then they perform the threshold decryption to obtain the output.

Intuitively, if a party aborts in the first round, then he is simply ignored. If he aborts in the second round, he is also ignored: since the other parties output flexible ciphertexts, these can be transformed to a public key representing the set of non-aborting parties. Those remaining parties can then proceed to perform the homomorphic computation. Finally, if a party aborts at the end, then it is too late – our threshold decryption algorithm only requires an honest majority of parties. We describe our three round protocol in Section 4.3.

Constructing TFHE. Our construction of TFHE is a distributed variant of the FHE scheme by Gentry, Sahai, and Waters [GSW13]. We inherit from their scheme that our TFHE does not need the evaluation keys that Asharov et al. required, which allows for a cleaner presentation. We outline some of the technical aspects of our construction here, after we recall the GSW construction. The public key in their construction is a matrix $\mathbf{B}$ and a vector $\vec{b} = \mathbf{B}\vec{s}+\vec{e}$ of the LWE form; the secret key is the LWE secret $\vec{s}$. To encrypt a bit m, the algorithm generates a random 0-1 matrix $\vec{R}$, and outputs $C = \mathsf{Flatten}\left(m \cdot I_D + \mathsf{BitDecomp}\big(\mathbf{R} \cdot \vec{b} \mid\mid \mathbf{R} \cdot \mathbf{B}\big)\right)$, where I_D is the identity matrix. (We will define $\mathsf{BitDecomp}$ and $\mathsf{BitDecomp}^{-1}$ in Section 2, but, essentially, these functions act as their names suggest, decomposing a field element into a binary representation, and building a field element from a binary string.) To decrypt, the algorithm takes row β (where roughly $2^\beta >$ some noise bound) and parses the row into $(C_{\beta,1}, C_{\beta,2}) \in \mathbb{Z}_q^\ell \times \mathbb{Z}_q^{n\cdot\ell}$. (The parameters ℓ, n, q will be set in the scheme. Here for exposition, we can omit them.) Then it outputs

$$\left\lfloor \frac{\mathsf{BitDecomp}^{-1}(C_{\beta,1}) - \langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \vec{s}\rangle}{2^\beta} \right\rceil.$$

The homomorphic evaluation of the GSW scheme is surprisingly simple and beautiful! For addition it is $C + C'$ and for multiplication $C \cdot C'$.

As we discussed, our TFHE does not immediately determine a public key with respect to all parties, as done by Asharov et al. [AJL+12]. Instead, we set the public parameter (CRS) to be $\mathbf{B}$, and let each party P_i output $\mathsf{pk}_i = \vec{b}_i = \mathbf{B}\vec{s}_i + \vec{e}_i$. Note that each $(\mathbf{B}, \vec{b}_i)$ is a GSW public key. The next challenge is how to generate flexible ciphertexts. A first natural idea would be: for P_i to generate a flexible ciphertext on some message m, P_i encrypts m under all GSW-type public keys $\{(\mathbf{B}, \vec{b}_i)\}_{[N]}$ to get $C = (C_1, \ldots, C_N)$. To transform C with respect to a set S, we simply output $\{C_i\}_{i\in S}$. However, this is not secure since it allows every party, independently, to decrypt P_i's ciphertext. A next idea would be P_i encrypts m under the key $(\mathbf{B}, \vec{b}_i)$ corresponding to his public key, and encrypts 0 for other keys $\{(\mathbf{B}, \vec{b}_j)\}_{j\neq i}$. The transform algorithm works the same. Intuitively, semantic security holds since P_i does not encrypt m under other people's keys. However, it is not clear how to jointly evaluate two ciphertexts from two parties, since the essential messages are encrypted under two different GSW public keys.

Our new idea to solve such challenge modifies C_j's for $j \neq i$: instead of generating $\mathsf{Enc}(0)$'s under $\{(\mathbf{B}, \vec{b}_i)\}_{[N]\setminus\{i\}}$, P_i outputs some *hints* for the transformation algorithm, but such hints will not hurt security. More specifically, we have the following design: P_i generates $C_i = \mathsf{Flatten}\left(m \cdot I_D + \mathsf{BitDecomp}\big(\mathbf{R} \cdot \vec{b}_i \mid\mid \mathbf{R} \cdot \mathbf{B}\big)\right)$ and $C_j = \mathsf{BitDecomp}\big(\mathbf{R} \cdot \vec{b}_j \mid\mid \mathbf{0}\big)$, for $j \neq i$, where the same $\mathbf{R}$ is used for all $\{C_j\}_{j\in[N]}$. Since each C_j only decreases the entropy of $\mathbf{R}$ by $|\mathbf{R} \cdot \vec{b}_j|$, we can still use a leftover-hash-lemma style approach to argue that m is hidden.

Then given a set S (including i), we can compute $C_S = \sum_{j \in S} C_j$. By unfolding the equation, we can see:

$$C_S = \left(m \cdot I_D + \mathsf{BitDecomp}\Big(\mathbf{R} \cdot \big(\sum_{j \in S} \vec{b}_j \big) \mid\mid \mathbf{R} \cdot \mathbf{B} \Big) \right),$$

which is of the form $\mathsf{Enc}(m)$ under the GSW public key $(\mathbf{B}, \sum_{j \in S} \vec{b}_j)$! This means any flexible ciphertext, after being transformed, results in an encryption under the GSW public key. Therefore, ciphertexts from different parties can be jointly computed after transformed to ones with respect to the same set S.

Our threshold decryption protocol needs to work for any set S of participants such that $|S| > \lceil N/2 \rceil$. So the parties should distribute the secret $\vec{s}_i$'s to all the other parties using a threshold secret sharing scheme. The challenging part is to design a *one-round* protocol. We use the fact that the decryption algorithm of the GSW scheme is essentially computing inner product (of a publicly known vector and the secret key), and Shamir's secret sharing scheme is highly compatible with inner product computation. In particular, each party P_i shares $\vec{s}_i$ into $(\vec{p}_i(1), \vec{p}_i(2), \ldots, \vec{p}_i(N))$ and sends $\vec{p}_i(j)$ to P_j, where $\vec{p}$ is a vector of polynomials for Shamir's shares. (See Appendix B for details of Shamir's secret sharing scheme). To compute $w = \langle \vec{u}, \sum_{j \in S} \vec{s}_j \rangle$ for some publicly known vector $\vec{u}$ (think of it as part of a ciphertext), each party can output $w_i = \langle \vec{u}, \sum_{j \in S} \vec{p}_j(i) \rangle$. Then it is not hard to see that these w_i's form shares of w, so after receiving a majority of shares each party can run the reconstruction without interaction!

Finally, we need to handle an additional technicality to deal with noise of evaluated ciphertexts, as pointed out by Asharov et al. [AJL$^+$12]. Intuitively, an evaluated ciphertext $\mathsf{Enc}(f(x))$ might contain noise that is related to the original input x, so we need to add additional smudging noise to eliminate any such link. In the decryption protocol of the work [AJL$^+$12], each party adds independent small noise to the output. However, this method will not work for our case because in our reconstruction procedure, these noise values are multiplied by the Lagrange coefficient, which can be too large. To solve this issue, we let each party P_i secret share some small noise η_i into $(r_i(1), \ldots, r_i(N))$ and send the shares to the other parties (where r_i is a random polynomial for the shares). Then each party P_i adds $\sum_{j \in S} r_j(i)$ to their output. By the linearity of the Shamir's sharing scheme, this is equivalent to adding $\sum_{j \in S} \eta_j$ to the original reconstructed output value. In Section 4.2 we go through this construction in detail. The new TFHE may be of independent interests.

1.3 Related Work

There is a long line of work studying the round complexity of secure computation, both in the semi-honest and malicious models, the two-party and multi-party settings, the honest majority and honest minority settings, and even in a variety of other models. We will not aim to survey all of this work, but mention what we know to be the best round complexity in the most relevant settings.

Constant round protocols have been known since Yao's original two-round construction for the two-party, semi-honest setting [Yao86], and Beaver et al.'s constant round protocol for the setting of a malicious minority [BMR90]. In the two-party, malicious setting, Katz and Ostrovsky give a five-round protocol and demonstrate that this is tight [KO04]. There are several works demonstrating constant round protocols in the multiparty, malicious majority setting ([KOS03, Pas04] Of course, with a malicious majority (including the two-party case), fairness is unachievable, so these results are in the security-with-abort model, and are not directly relevant to our own work.

In the multiparty setting with a malicious minority, the best known construction is the two-round protocol by Garg et al. [GGHR14], but, as we outlined above, their result does not ensure fairness. For $t < N/5$ corruptions, Damgård and Ishai give a three-round protocol with a guarantee of output delivery [DI05], though they require private point-to-point channels, and establishing these would add at least one additional round. For $t < N/2$, the exact round complexity of their protocol is a bit hard to discern, but it is greater than four (and we believe more); in this domain, the five-round protocol of Asharov et al. [AJL$^+$12], which also guarantees output delivery, is the best known.

In the semi-honest, two-party setting, Yao's original construction already achieves two-rounds.

2 Preliminaries

In this section, we present basic vector operations. We describe the security definitions for MPC in Section A and the LWE assumptions in Section C.

2.1 Elementary Vector Operations

We define a number of vector/matrix operations that we describe below. Let $\vec{a}, \vec{b}$ be vectors of dimension k. Let $\ell = \lfloor \log q \rfloor + 1$ for some modulus q. Note that the operations we describe are also defined over matrices, operating *row by row* on the matrix, and that all arithmetic is over $\mathbb{Z}_q$.

$\mathsf{BitDecomp}(\vec{a})$ = the $k \cdot \ell$ dimensional vector $(a_{1,0}, \ldots, a_{1,\ell-1}, \ldots, a_{k,0}, \ldots a_{k,\ell-1})$ where $a_{i,j}$ is the j^{th} bit in the binary representation of a_i, with bits ordered from least significant to most significant.

$\mathsf{BitDecomp}^{-1}(\vec{a}')$ For $\vec{a}' = (a_{1,0}, \ldots, a_{1,\ell-1}, \ldots, a_{k,0}, \ldots a_{k,\ell-1})$, let
$\mathsf{BitDecomp}^{-1}(\vec{a}') = \left(\sum_{j=0}^{\ell-1} 2^j a_{1,j}\ , \ldots,\ \sum_{j=0}^{\ell-1} 2^j a_{k,j}\right)$, but defined even when $\vec{a}'$ isn't binary.

$\mathsf{Flatten}(\vec{a}')$ $= \mathsf{BitDecomp}\left(\mathsf{BitDecomp}^{-1}(\vec{a}')\right)$

$\mathsf{Powersof2}(\vec{b})$ $= (b_1, 2b_1, 4b_1, \ldots, 2^{\ell-1}b_1, \ldots, b_k, \ldots 2^{\ell-1}b_k)$.

3 Impossibility Result

In this section, we are going to show that it is impossible to construct a two-round secure protocol for general multi-party computation with fairness, even with an honest majority of players. Our impossibility results holds in the *standalone* model, even with non-rushing fail-stop adversaries and CRS. We assume that the players have both point-to-point channels and a public broadcast channel, but they do not have private point-to-point channels – an eavesdropper can listen to all channels.[3] We note that our three round protocol from Section 4 can be collapsed into a three round protocol if we give the users access to a PKI of the appropriate form, so the assumption of non-private channels in our lower-bound is natural.[4] A more formal proof follows.

Theorem 3.1 *Let $\mathcal{C}$ be a family of circuits, and let Π be a polynomial-time, 2-round, 3-party secure protocol for computing $U(C, x, 0) = C(x)$ for any $C \in \mathcal{C}$, with fairness in the standalone model. Then there exists a virtual black-box obfusctor for general circuits as Definition C.2.*

Proof: We describe a VBB obfuscator $\mathcal{O}$ for all circuits in $\mathcal{C}$ (as Definition C.2). Before doing that, we define some notation and our next message functions. We let $\mathcal{M}$ denote the set of valid messages in the secure computation. We let $\perp \in \mathcal{M}$ denote a special abort symbol, and we let $\emptyset$ denote the empty transcript (before any messages have been sent). A *partial incoming transcript*, is either $\emptyset$ (if no messages have been sent yet), or of the form $(\mathcal{M}, \mathcal{M})$, where each message is received from one of the two other parties in the first round of the protocol. A *partial outgoing transcript* is of the same form, but represents the two messages sent by a single party in the first round, each going to one of the other parties. A *full incoming transcript* is of the form $((\mathcal{M}, \mathcal{M}), (\mathcal{M}, \mathcal{M}))$, where the first pair of messages are those received in the first round, and the second pair are those received in the second round. We define the following set of circuits.

$\pi_{\mathsf{i,j}}(x, \tau, r)$: for parties $i, j \in \{1, 2, 3\}$, on input value x, partial incoming transcript τ and randomness r, the circuit outputs i's next message to j.
$\pi_{\mathsf{out}}(x, \tau, r_2)$: the circuit computes P_2's output in the secure computation, given input x, full incoming transcript τ and randomness r_2.

[3]Our lower bound holds even when the eavesdropper only listens to some channels.
[4]Although we allow the eavesdropping adversary to corrupt two private channels at once, we do not allow it to corrupt the parties themselves, so we do still maintain an honest majority. However, there is still room to consider a weaker model where the eavesdropper can only listen to a single channel.

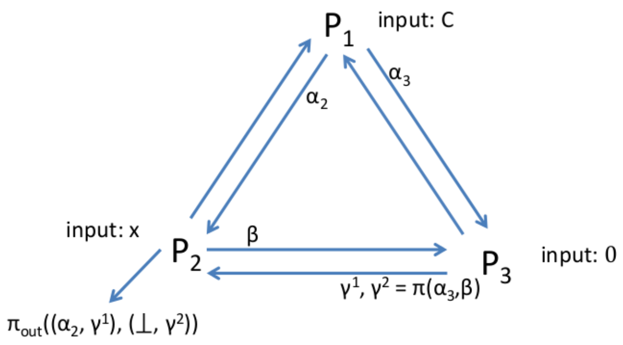

Figure 1: A depiction of the messages used in the circuit $\Gamma_{\alpha_2,\alpha_3}$. α messages are sent by party P_1, β messages by P_2, and γ messages by P_3. A subscript i indicates that the recipient is P_i, and a superscript indicates a round number. Since we do not need all protocol messages, we drop subscripts and superscripts where we can.

The VBB obfuscation of circuit C is as follows. $\mathcal{O}(C)$ chooses randomness r_1 and computes $\alpha_2 = \pi_{12}(C, \emptyset, r_1)$, $\alpha_3 = \pi_{13}(C, \emptyset, r_1)$. Note that they are the first-round messages from P_1 to P_2 and P_3. Then the obfuscator outputs the following circuit $\Gamma_{\alpha_2,\alpha_3}(x; r_2, r_3)$, which, on input x and randomness r_2, r_3 , performs the following computations:

- $\gamma^{(1)} = \pi_{32}(0, \emptyset, r_3)$; $\beta = \pi_{23}(x, \emptyset, r_2)$. (The relevant first round messages.)
- $\gamma^{(2)} = \pi_{32}(0, (\alpha_3, \beta), r_3)$. (The relevant second round message.)
- Output $\pi_{\mathsf{out}}(x, \left((\alpha_2, \gamma^{(1)}), (\perp, \gamma^{(2)})\right), r_2)$. ($\mathsf{P}_2$'s output, given his full incoming transcript.)

Basically, the circuit simulates $\mathsf{P}_2, \mathsf{P}_3$'s messages when P_1 sends out α_2, α_3 and then aborts in the second round.

We claim that for any $C \in \mathcal{C}$, $\Gamma_{\alpha_2,\alpha_3}$ is a secure VBB obfuscation of C. Efficiency of $\Gamma_{\alpha_2,\alpha_3}$ follows from the fact that the secure computation is polynomial-time. Correctness follows from the fairness of the underlying secure computation protocol – we note that by the correctness of the protocol, P_1 can learn the output after he sees all the incoming messages, regardless of whether or not he aborts in the second round. Thus by fairness of the protocol, P_2 should also receives the output, regardless of whether or not P_1 aborts in the second round. Thus, given the transcript $(x, \left((\alpha_2, \gamma^{(1)}), (\perp, \gamma^{(2)})\right)$, P_2 can compute the output.

To prove the VBB property, recall that we need to prove that for any adversary $\mathcal{A}_{\mathcal{O}}$, for any circuit C, there exists a simulator $\mathcal{S}_{\mathcal{O}}$ and a negligible function ϵ such that

$$|\Pr[\mathcal{A}_{\mathcal{O}}(\Gamma_{\alpha_2,\alpha_3}) = 1] - \Pr[\mathcal{S}_{\mathcal{O}}^{C}(1^{|C|}) = 1]| < \epsilon(|C|)$$

By the security of the underlying secure computation, we know there exists an ideal-world simulator, which we will denote by $\mathcal{S}_{\mathsf{E}}$, that simulates the view of an eavesdropper who listens to the channels between P_1 and P_2, and between P_1 and P_3. We will denote by $\mathcal{S}_{\mathsf{E}}^{(1)}$ the result of running $\mathcal{S}_{\mathsf{E}}$ and restricting the output to the partial outgoing transcript, i.e. the first round messages sent from P_1. Then, $\mathcal{S}_{\mathcal{O}}$ gets $(\tilde{\alpha}_2, \tilde{\alpha}_3) \leftarrow \mathcal{S}_{\mathsf{E}}^{(1)}$, and constructs the circuit $\Gamma_{\tilde{\alpha}_2,\tilde{\alpha}_3}$, as described above; we note that neither C nor r_1 are needed, once $\tilde{\alpha}_2$ and $\tilde{\alpha}_3$ are computed. Finally, $\mathcal{S}_{\mathcal{O}}$ outputs $\mathcal{A}_{\mathcal{O}}(\Gamma_{\tilde{\alpha}_2,\tilde{\alpha}_3})$.

Suppose that this does not meet the above security requirement. It follows that there exists a distinguisher $\mathcal{D}$ that distinguishes between a real world execution of the protocol and the ideal simulation of $\mathcal{S}_{\mathsf{E}}$. This follows immediately, because the only difference between the true obfuscation and the simulated obfuscation is the way in which α_2 and α_3 are generated. Therefore, on input transcript τ, $\mathcal{D}$ simply takes the messages α_2, α_3 that constitute the first round messages sent from P_1 to P_2 and P_3 respectively, and he completes the construction of $\Gamma_{\alpha_2,\alpha_3}$ himself. $\mathcal{D}$ then runs $\mathcal{A}_{\mathcal{O}}$ on the resulting circuit and determines from the output whether τ was simulated.

Remark. The lower bound proof can be extended to rule out protocols in the CRS model, using the same idea. The obfuscated circuit will now embed crs as a common reference string, and in the security proof, the simulator will simulate the string, i.e. $(\tilde{\mathsf{crs}}, \tilde{\alpha}_2, \tilde{\alpha}_3) \leftarrow \mathcal{S}_{\mathsf{E}}^{(1)}$. ■

Two Round Feasibility with a PKI. Note that the proof breaks down if the parties have access to private channels, including in the scenario where they have access to a PKI. This is because we need *both* first-round messages sent from P_1 in order to simulate round two of the protocol. In particular, without access to α_3, we could not correctly simulate $\gamma^{(2)}$ (as sent from P_3 to P_2 in round two), and therefore we could guarantee the correct output of the obfuscation. The only way to gain access to both α_2 and α_3 is either by eavesdropping on multiple channels, or by corrupting two parties, but this latter approach would violate our assumption of an honest majority. Indeed, as we mentioned previously, and as we will see later, our construction in Section 4 can be collapsed to two rounds if we have access to a PKI, with public keys of a particular form. It is still an open question whether a two-round protocol with guaranteed output delivery is possible, given access only to private channels.

4 Towards Fairness and Guarantee of Output Delivery

The previous section shows that two-round fair protocols are in general impossible. As discussed in the introduction, we can construct a three-round fair protocol by adding one more round to the protocol by Garg et al. [GGHR14]; yet it is unclear how to construct three-round protocols with guarantee of output delivery. In this section, we present our main contribution – we construct a new threshold FHE scheme, which extends the notion of threshold FHE by Asharov et al. [AJL+12] with enriched features. We elaborate on these below.

4.1 New Threshold Fully Homomorphic Encryption Scheme

As discussed in the introduction, our TFHE introduces a new idea of *flexible* and *transformed* ciphertexts that play an important role in our 3-round MPC construction. Here we first present the syntax: a threshold fully homomorphic encryption scheme (TFHE) is basically a homomorphic encryption scheme, with the difference that the key generation and decryption are N-party protocols instead of algorithms. We will consider protocols defined in terms of some common parameter pp.

- $\mathsf{TFHE.Gen}(\mathsf{pp})$ **(Key Generation Protocol.)** Initially each party holds some parameter pp. At the conclusion of the protocol, each party P_i for $i \in [N]$ publishes a public key pk_i, and keeps a private key sk_i.

- $\mathsf{TFHE.Dec}_S(C; \vec{v})$ **(Threshold Decryption Protocol.)** Let S be a set in $[N]$, and $\vec{v} = \{v_i : i \in S\}$ be some secret values, each held by one party. The protocol is run among parties $\{P_i : i \in S\}$. Initially each party holds a secret input $\vec{v}[i]$, a secret key sk_i, and receives a ciphertext C as the public input. At the end, the parties in the set can compute the decrypted message m. Intuitively, the secret input $\vec{v}$ is used for smudging the noise.

  Note: in the setting with honest majority, we assume that $|S| \geq [N/2] + 1$. For simplicity, we assume the input ciphertext C has already been transformed to one that corresponds to the set of public keys $\mathsf{pk}_S = \{\mathsf{pk}_i : i \in S\}$. See the syntax below for further exposition.

- $\mathsf{TFHE.Enc}_i(\mathsf{pp}, \mathsf{pk}_1, \ldots, \mathsf{pk}_N; m)$ **(Encryption Algorithm.)** Let parties $\{\mathsf{P}_i\}_{i\in[N]}$ participate in the protocol, and $\{\mathsf{pk}_i\}_{i\in[N]}$ be the set of their public keys. The encryption algorithm is non-interactive and run by party P_i. The algorithm takes inputs the public parameter, the public keys $\{\mathsf{pk}_i\}_{i\in[N]}$, a message m, and computes a ciphertext C.

  We implicitly require that the ciphertexts here are **flexible** in the sense that they do not *commit* to a particular public key/secret key yet; in particular, we can use the algorithm below to transform a flexible ciphertext into one that corresponds to a set of public keys.

- $\mathsf{TFHE.TransCT}(C; S)$ **(Ciphertext Transform Algorithm.)** The algorithm takes inputs a *flexible* ciphertext C (from the above encryption algorithm), and a set $S \subseteq [N]$ and outputs a *transformed* ciphertext C_S. The ciphertext can be thought as one under the set of joint keys: $\mathsf{pk}_S = \{\mathsf{pk}_i : i \in S\}$.

- $\mathsf{TFHE.Eval}(f, C_1, \dots, C_t; S)$ **(Evaluation Algorithm.)** The evaluation algorithm is non-interactive. A party P_i (can be any party) receives inputs a function $f : \{0,1\}^t \to \{0,1\}$, flexible ciphertexts $C_1, \dots, C_t$, and a set $S \subseteq [N]$. He computes an evaluated ciphertext C'_S with respect to the set S, which can be thought as an evaluated ciphertext under the joint public key pk_S defined as above.

We summarize the main differences between our TFHE and that of the prior work [AJL+12].

1. Our key generation does not output a *joint* public key. Instead each party will only output their own public key pk_i. Then parties can run the Eval algorithm to homomorphically compute on the ciphertexts under pk_S for some set S decided later. As pointed out in the introduction, this is an important feature.

2. We do not require evaluation keys.

3. The construction of the prior work requires all parties to participate in the decryption protocol (in the non-interactive case). Here we allow a subset of parties to run the protocol; moreover, we allow a "threshold" type of decryption where a majority of parties can decrypt the ciphertext.

These new features play an important role: intuitively, when a party generates a ciphertext, he does not know who else might abort. The flexibility of ciphertexts handles this problem – the parties can generate ciphertexts first, and later on decide a set of public key (namely $\mathsf{pk}_S = \{\mathsf{pk}_i : i \in S\}$), so that the flexible ciphertexts can be transformed with respect to pk_S. Then the parties can perform homomorphic computation with respect to pk_S and run the threshold decryption algorithm.

Similar to the work [AJL+12], we do not define the security of TFHE on its own. The reason is similar: 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 4.3.

4.2 Construction of Our New TFHE

Following the intuition in the introduction, we describe our construction.

Common Parameter. All parties receive the common parameter pp of the form: let N be the number of parties, $L = \mathrm{poly}(\kappa)$ be the maximum depth of the circuits supported by the TFHE evaluation algorithm. Then we choose a modulus q of $\mathrm{poly}(L, N)$ bits, lattice dimension parameter $n = n(L, N)$, and error distribution $\chi = \chi(\kappa, L, N)$ appropriately for LWE security against 2^κ known attacks. Also, choose parameter $m = m(\kappa, L) = O((n+N)\log q)$. Let the distribution χ be B_χ-bounded (i.e. with overwhelming probability, a sample from χ has the absolute value less than B_χ). Let $\ell = [\log q] + 1$, $D = (n+1) \cdot \ell$, and $B_{\mathsf{smug}} \in \mathbb{Z}$ be an integer bound, satisfying the following relations:

$$\frac{(D+1)^L \cdot N \cdot B_\chi}{B_{\mathsf{smug}}} = \mathsf{negl}(\kappa),\ \ B_{\mathsf{smug}} < q/8.$$

Then $\mathsf{pp} = (n, q, \chi, m, B_\chi, B_{\mathsf{smug}}, \mathbf{B})$ where $\mathbf{B}$ is sampled uniformly from $\mathbb{Z}_q^{m \times n}$.

$\mathsf{TFHE.Gen}(\mathsf{pp})$**:** This is a two-round protocol among N parties.

- **(Round 1):** Each party P_i samples a random vector $\vec{s}_i \in \mathbb{Z}_q^n$, and computes $\vec{b}_i = \mathbf{B} \cdot \vec{s}_i + \vec{e}_i$ where $\vec{e}_i \leftarrow \chi^m$. Then P_i broadcasts $\mathsf{pk}_i = \vec{b}_i$, and keep $\vec{s}_i$ secretly.

- **(Round 2):** Each party P_i secret shares $\vec{s}_i$ using the Shamir Secret Sharing Scheme with threshold $[N/2] + 1$. Let $\vec{p}_i$ denote the random polynomial vector (of degree $[N/2] + 1$) generated by P_i where $\vec{p}_i(0) = \vec{s}_i$ (This is how the Shamir Secret Sharing works). P_i sends $\vec{p}_i(j)$ to P_j for $j \in [N]$. At the end, P_i sets $\mathsf{sk}_i = (\vec{p}_1(i), \vec{p}_2(i), \dots, \vec{p}_N(i))$.

Note that although we do not assume secure point-to-point channels, sending private message in the second round is achievable – everyone can send a public key in the first round, and later on every party encrypts the outgoing messages. For simplicity, we just assume there are secure point-to-point channels available in the second round.

$\mathsf{TFHE.Dec}_S(C; \vec{v})$**:** Let $\vec{v}$ be a vector of $|S|$ numbers (error terms), where $\vec{v}[i]$ (the element indexed by i) is held by party P_i for $i \in S$; let C be a ciphertext. For $S \subseteq [N]$ such that $|S| \geq N/2$, this is a one-round protocol among parties $\{\mathsf{P}_i : i \in S\}$. For simplicity, we assume that C is a transformed ciphertext that corresponds to pk_S.

- Each party P_i parses C as a matrix in $\mathbb{Z}_q^{D\times D}$. Then he picks the β-th row, C_β, where $\beta = \lfloor \log_2(q/2) \rfloor$. Note that $2^\beta \in (q/4, q/2]$. Then parse $C_\beta = (C_{\beta,1}, C_{\beta,2})$ where $C_{\beta,1} \in \mathbb{Z}_q^{\ell}$, $C_{\beta,2} \in \mathbb{Z}_q^{n\cdot\ell}$. Then he computes $\vec{z}_i = \sum_{j\in S} \vec{p}_j(i)$ and broadcasts $w_i = \langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \vec{z}_i \rangle + \vec{v}[i]$.
- At the end, each party picks an arbitrary subset $T \subseteq S$ such that $|T| = [N/2] + 1$. Then they compute $w = \sum_{k\in T} \mu_k(0) w_k$, where μ_k is the Lagrange polynomial. Finally they output $\left\lfloor \frac{\mathsf{BitDecomp}^{-1}(C_{\beta,1}) - w}{2^\beta} \right\rceil$.

$\mathsf{TFHE.Enc}_i(\mathsf{pp}, \mathsf{pk}_1, \ldots, \mathsf{pk}_N; m)$**:** This is the i-th party's encryption algorithm. Let $m \in \{0, 1\}$ be the input message.

- The algorithm parses pp as a matrix $\mathbf{B} \in \mathbb{Z}_q^{m\times n}$, $\mathsf{pk}_j = \vec{b}_j \in \mathbb{Z}_q^m$ for $j \in [N]$. Then it samples a random matrix $\mathbf{R} \in \{0,1\}^{D\times m}$, and computes $\mathbf{W}_j = \mathsf{BitDecomp}\big(\mathbf{R} \cdot \vec{b}_j \mid\mid 0^{D\times n}\big)$ for $j \neq i$. It computes $\mathbf{W}_i = \mathsf{Flatten}\Big(m \cdot I_D + \mathsf{BitDecomp}\big(\mathbf{R} \cdot \vec{b}_i \mid\mid \mathbf{R} \cdot \mathbf{B}\big)\Big)$, where I_D is the identity matrix of dimension $D \times D$. It outputs $C = (\mathbf{W}_1, \ldots, \mathbf{W}_N)$.

$\mathsf{TFHE.TransCT}(C; S)$**:**

- The algorithm parses C as a N matrices $(\mathbf{W}_1, \ldots, \mathbf{W}_N)$. It outputs
$C_S = \sum_{j\in S} \mathbf{W}_j$.

$\mathsf{TFHE.Eval}(f, C_1, \ldots, C_t; S)$**:**

- For simplicity, we assume that all the ciphertexts $C_1, \ldots, C_t$'s are transformed to ones that correspond to pk_S (otherwise we can apply the above $\mathsf{TFHE.TransCT}$ first). We then observe that actually a transformed ciphertext is of the same form of the GSW scheme [GSW13] where the public key is $(\sum_{k\in S} \vec{b}_k \mid\mid \mathbf{B})$. Thus, we can run exactly the same evaluation as the GSW scheme! More specifically, we represent f as a circuit (with all NAND gates). Then we can homomorphically compute $NAND(C, C')$ by outputting $\mathsf{Flatten}(I_D - C \cdot C')$. See the work [GSW13] for detailed explanation.

With out setting of parameters, we can argue that flexible ciphertexts do not leak the underlying messages to the other parties (and so do the transformed ciphertexts, since they can be obtained deterministically from flexible ciphertexts). This can be shown formally using the lemma below in a strait-forward way as done by the work [GSW13]. See their work [GSW13] for further exposition[5].

Lemma 1 (Implicit in [Reg05]) *Let n, m, χ, q be parameters such that the $LWE_{n,q,\chi}$ holds, and N be some polynomial. Then for $m = O((n+N)\log q)$, for any vectors $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_{N-1} \in \mathbb{Z}_q^m$, then the distribution described as above $(\mathbf{B}, \vec{b}, \mathbf{R}\cdot(\mathbf{B}||\vec{b}), \mathbf{R}\cdot(\vec{b}_1|| \ldots ||\vec{b}_{N-1}))$ is computationally indistinguishable from $(\mathbf{B}, \vec{u}, \mathbf{U}, \mathbf{R}\cdot(\vec{b}_1|| \ldots ||\vec{b}_{N-1}))$, where $\mathbf{B}$ is uniform over $\mathbb{Z}_q^{m\times n}$, $\vec{u}$ is uniform over $\mathbb{Z}_q^m$, $\mathbf{U}$ is uniform over $\mathbb{Z}_q^{D\times(n+1)}$, and $\mathbf{R}$ is uniform over $\{0,1\}^{D\times m}$, $D = (n+1)\cdot\ell$, $\ell = [\log q] + 1$.*

[5]Our setting of parameters is slightly different from that of the work [GSW13], so our parameters in the lemma are slightly different. The analysis is essentially identical.

4.3 Three-round MPC with Guarantee of Output Delivery

Now we are ready to present our new three-round MPC for general functions using the new TFHE we have developed in the previous section. We first present a simpler case that considers MPC for polynomial-time deterministic boolean function f (where all parties receive the same bit). Moreover, the security holds against static *semi-malicious* fail-stop attackers[6] corrupting less than half of the parties. In Section 5, we discuss how to handle general cases using standard techniques.

Remark 4.1 *Our protocol only needs a public broadcast channel[7]. For simplicity of presentation, we make the following two assumptions. First, there are secure point-to-point channels available. Second, when a party distributes shares to the other parties, he must either send messages to all parties or send messages to no one. These assumptions are not necessary, and we sketch how to achieve them in our protocol using the broadcast channel. We observe that our protocol will only use the secure channels to distribute shares in the second round. So in the first round everyone can publish a public key, and then in the second round, everyone broadcasts encryptions of the shares (under different parties' public keys). This can implement the secure channels, and ensure that parties will either abort (not broadcast at all) or distribute messages to all the other parties.*

Our Construction. Let $f : \{0,1\}^{(\ell_{\mathsf{in}})^N} \to \{0,1\}$ be a function computed by a depth L circuit, where ℓ_{in} is the input length of each party.

Input: Each party P_i holds some input $x_i \in \{0,1\}^{\ell_{\mathsf{in}}}$. The parties share the public parameter pp as described in the TFHE scheme. (pp can be viewed as the common reference string. The generation of pp depends on L, since we need the TFHE to support circuits up to depth L).

The Protocol:

- **Round 1:** The parties execute the first round of the $\mathsf{TFHE.Gen}(\mathsf{pp})$. If anyone aborts in this round, then he is simply ignored. Let $S_1 \subseteq [N]$ be the set of non-aborting parties at this round. At the end of this round, each party holds all $\{\mathsf{pk}_i\}_{i \in S_1}$.

- **Round 2:** The parties execute the following procedures at the same time:

  - The (currently non-aborting) parties execute the second round of the $\mathsf{TFHE.Gen}(\mathsf{pp})$.
  - For $i \in S_1$, P_i broadcasts an encryption of his input using the algorithm $\mathsf{TFHE.Enc}_i(x_i)$ (encrypt it bit-by-bit). Note that these are a flexible ciphertexts.
  - Each P_i samples a uniformly random error term from $\eta_i \leftarrow [-B_{\mathsf{smug}}, B_{\mathsf{smug}}]$, and compute random Shamir secret shares (with the same threshold $[T/2]+1$). Denote the polynomial as r_i (note that $r_i(0) = \eta_i$). Then each P_i sends $r_i(j)$ to party P_j for $j \neq i$.

  Let $C_i = (C_{i,1}, C_{i,2}, \ldots, C_{i,\ell_{\mathsf{in}}})$ be the broadcasted ciphertexts from P_i, and $(r_i(1), \ldots, r_i(N))$ be the shares from P_i to the other parties.

  If anyone aborts at this round, either not sending the second round of $\mathsf{TFHE.Gen}(\mathsf{pp})$, the ciphertexts, or the shares of error terms, then he (and his input) are again ignored. Let $S_2 \subseteq S_1$ be the set of non-aborting parties.

- **Round 3:** Now each non-aborting party in S_2 first transforms the ciphertexts he received to ones that correspond to pk_{S_2}. Let $\{C_{j,k}\}_{j \in S_2, k \in [\ell_{\mathsf{in}}]}$ be the broadcasted ciphertexts. For $i \in S_2$, P_i first computes $C_{j,k}^{S_2} = \mathsf{TFHE.TransCT}(C_{j,k}; S_2)$ for $j \in S_2$, $k \in [\ell_{\mathsf{in}}]$.

  Let f^{S_2} be the residual function where the inputs of $[N] \setminus S_2$ are replaced with the default values. P_i homomorphically computes the residual function, i.e. $C^* = \mathsf{TFHE.Eval}(f^{S_2}, \{C_{j,k}^{S_2}\}_{j \in S_2, k \in [\ell_{\mathsf{in}}]})$.

[6]Basically, a semi-malicious attacker is one whose behavior follows the protocol with *some* input and randomness he must know. Protocols that achieve such security can be upgraded to malicious security without adding a coin flipping round (c.f. Section 5). See Section A.1 for further details about the notion and its advantage.

[7]In the semi-malicious setting, this can be easily implemented by reliable public point-to-point channels, where an eavesdropper can listen to the all channels but cannot modify the messages

Then each P_i computes $v_i = \sum_{k \in S_2} r_k(i)$. Finally, they run the threshold decryption $\mathsf{TFHE.Dec}_{S_2}(\{C^*; \vec{v}_{S_2}\})$, where $\vec{v}_{S_2}$ denotes the vector of the following set $\{v_j : j \in S_2\}$.

Recall that the protocol $\mathsf{TFHE.Dec}$ handles situations when parties abort. In this round, parties broadcast some messages, and a majority of them is sufficient to recover the output.

Theorem 4.2 *Let f be any deterministic functionality with N inputs and one output. Let pp be parameters sampled according to the choice as the TFHE above, and the corresponding LWE assumption holds. Then the above protocol π UC-realizes the ideal functionality $\mathcal{F}_f$ with guarantee of output delivery, in the presence of any static (semi-malicious) fail-stop adversary who corrupts less than $[N/2]$ parties.*

As explained in the introduction, the transformed ciphertexts $\{C^{S_2}_{j,k}\}_{j \in S_2, k \in [\ell_{\mathsf{in}}]}$ are GSW ciphertexts under the public key $(\mathbf{B}, \sum_{i \in S_2} \vec{b}_i)$. Therefore, by applying the evaluation algorithm, C^* is a ciphertext of the output y. Each party in our threshold decryption protocol, as explained, outputs a share of y by computing some inner product with the shares (and substraction). Thus, the correctness holds.

To prove security, we need to construct a simulator $\mathcal{S}$ that generates the views of the honest parties. We sketch the construction: the simulator simulates the public parameter faithfully, and generates the messages in each round as follows. Let $\mathcal{I}$ be the set of corrupted set.

- **(First round).** $\mathcal{S}$ simulates the public keys of honest parties' by random vectors $\vec{u}_i$ for $i \notin \mathcal{I}$.
- **(Second round).** $\mathcal{S}$ simulates the encrypted ciphertexts by $\mathsf{TFHE.Enc}(0)$, and simulates the error terms and shares of secret keys by sending random values (or vectors).
- **(Third round).** $\mathcal{S}$ then reads the witness tapes of the adversary to get secret keys and inputs from the corrupted parties. He sets the aborting parties' inputs to be the default value, and then queries the ideal functionality to receive the output y. From the output y and the secret keys of the corrupted parties, $\mathcal{S}$ then figures out consistent outputs of the honest parties.

Intuitively, the LWE assumption guarantees that the simulation in round 1 is indistinguishable, and Lemma 1 guarantees that $\mathsf{TFHE.Enc}(0)$ is indistinguishable form the encryptions in the real world. The last step is the most challenging, and we will further explain the ideas in the appendix.

In Section E, we present the detailed analyses of correctness and security with further exposition.

5 Variants and Generalizations

In this section, we discuss variants of our basic protocol in the following aspects: (1) how to handle functionalities with longer inputs, (2) how to handle randomized functionalities, (3) how to compile a protocol that is secure against semi-malicious adversaries into one that is secure against malicious adversaries, and (4) how to reduce one round by using a PKI setup. These issues can be handled using standard techniques as presented in the work of Asharov et al. [AJL+12]. We highlight the ideas and refer curious readers to their work for further details.

Functions with longer outputs. Let $f : \{0,1\}^{(\ell_{\mathsf{in}})^N} \to \{0,1\}^{\ell_{\mathsf{out}}}$ be an N-ary functionality. We consider ℓ_{out} boolean functionalities $\left\{f_i : \{0,1\}^{(\ell_{\mathsf{in}})^N} \to \{0,1\}\right\}_{i \in [\ell_{\mathsf{out}}]}$ where each f_i outputs the i-th bit of f. Let π_i be the protocol computing f_i as we described in Section 4. To compute f, we simply run $\pi_1, \dots, \pi_{\ell_{\mathsf{out}}}$ in parallel, and we treat an abort in any one of the execution as an abort in all executions. To argue that the resulting protocol is secure against an arbitrary semi-malicious adversary, we also require the adversary to include proofs, in the form of witnesses written to their witness tape, of input-consistency across the parallel executions. This is to enforce that the adversary is using the same inputs for all the subprotocols. Below we will describe a compiler that upgrades the protocol to one against malicious adversaries.

Randomized functionalities. Our basic MPC protocol only considers deterministic functionalities where all the parties receive the same output. It can be generalized to handle with randomized functionalities and individual outputs via a standard transformation. Basically in this transformation, instead of computing some randomized function $f(x_1, \dots, x_N; r)$, the parties compute the deterministic function $f'((x_1, r_1), \dots, (x_N, r_N)) = f\left(x_1, \dots, x_N; \oplus_{i \in [N]} r_i\right)$. This transformation does not add additional rounds.

Semi-malicious security to malicious security. Our basic MPC protocol is only secure in the semi-malicious setting. Asharov et al. [AJL+12] presents a simple and general round-preserving compiler from semi-malicious to fully malicious security using UC NIZKs [DDO+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 particular, we present the theorem of Asharov et al. [AJL+12]:

Theorem 5.1 ([AJL+12]) *There is a generic round-preserving compiler such the following holds. Let $\mathcal{F}$ be an N-ary functionality and π be an N-party protocol. Suppose π t-securely computes F against semi-malicious fail-stop adversaries with guarantee of output delivery (or fairness), then the compiled protocol π' t-securely computes $\mathcal{F}$ against malicious adversaries with guarantee of output delivery (or fairness, respectively) in the CRS, F_{ZK}, and authenticated broadcast-hybrid model. Moreover, π' has the same round complexity as π.*

Together with Theorem 4.2, we are able to achieve the following corollary:

Corollary 5.1 *Assume that the LWE assumption holds and UC-NIZK exists. Then there exists a three-round MPC in the CRS and authenticated broadcast hybrid model, with a guarantee of output delivery, and providing security against a malicious adversary that corrupts less than half of the parties.*

Two rounds with PKI. We recall that in the first round of our protocol, each party just publishes some public key $\vec{b}_i = \mathbf{B} \cdot \vec{s}_i + \vec{e}_i$, which is independent of the input. If there is an additional setup *public-key infrastructure* (PKI), then we can move the first round to the PKI. Thus the entire MPC execution would consist only of the remaining two rounds. The resulting PKI is very simple and does not require a trusted party for setup; 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 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). In the malicious case, the parties need to provide a zero-knowledge proof of knowing some randomness of their public keys registered in the PKI. This is similar to our original protocol (without PKI) where the parties need to provide a zero-knowledge proof of knowing some randomness of their first round messages.

References

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, April 2012.

[AJW11] Gilad Asharov, Abhishek Jain, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. Cryptology ePrint Archive, Report 2011/613, 2011. `http://eprint.iacr.org/2011/613`.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, August 2001.

[BH15] Mihir Bellare and Viet Tung Hoang. Adaptive witness encryption and asymmetric password-based cryptography. In *PKC*, 2015.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CL14] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 466–485. Springer, December 2014.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 364–369, 1986.

[DDO$^+$01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, August 2001.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 378–394. Springer, August 2005.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 74–94. Springer, February 2014.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, August 2013.

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 335–354. Springer, August 2004.

[KOS03] Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Round efficiency of multi-party computation with a dishonest majority. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 578–595. Springer, May 2003.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In László Babai, editor, *36th ACM STOC*, pages 232–241. ACM Press, June 2004.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 333–342. ACM Press, May / June 2009.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Security Definitions for Multi-party Computation

In this section, we present security definitions of secure computation with fairness and secure computation with guarantee of output delivery. We work in both the standalone model and the standard universal composability (UC) framework of Canetti [Can01]. We present a summary of the UC framework from the work [AJW11], and refer readers to [Can01] for further details.

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 $\tilde{\mathsf{P}}_1, \ldots, \tilde{\mathsf{P}}_N$ 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 $\mathsf{P}_1, \ldots, \mathsf{P}_N$ and a real-world adversary $\mathcal{A}$, 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.

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

Definition A.1 (UC Security [Can01]) *We say a protocol π securely realizes $\mathcal{F}$ if for any PPT adversary $\mathcal{A}$ in the real world, there exists a PPT simulator $\mathcal{S}$ in the ideal world, so that no PPT environment $\mathcal{Z}$ is able to tell the real world execution from the ideal world execution, i.e., $\{\boldsymbol{Real}_{\pi,\mathcal{A},\mathcal{Z}}(1^\kappa, z)\}_{\kappa\in\mathbb{Z}, z\in\{0,1\}^*} \approx \{\boldsymbol{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa, z)\}_{\kappa\in\mathbb{Z}, z\in\{0,1\}^*}$.*

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

UC and Standalone model. In the standalone security, the environment machine $\mathcal{Z}$ (i.e., the distinguisher) provides inputs to all protocol participants and the adversary at the beginning of protocol execution, and it receives outputs from these entities when the execution is complete. The environment and the adversary are not allowed to communicate during the protocol execution. Protocols secure in the standalone security model can be composed sequentially. On the other hand, in the UC framework, the environment and the adversary are always allowed to communicate. Protocols secure in the UC framework can be composted with arbitrary protocols. It is obvious that UC security implies stand-alone security.

A.1 Adversarial behavior

Loosely speaking, the aim of a secure multiparty protocol is to protect the honest parties against dishonest behavior from the corrupted parties. This is normally modeled using a central adversarial entity, which controls the set of corrupted parties and instructs them how to operate. That is, the adversary obtains the views of the corrupted parties, consisting of their inputs, random tapes and incoming messages, and provides them with the messages that they are to send in the execution of the protocol.

We differentiate between three types of adversaries:

- **Semi-honest adversaries:** a semi-honest adversary always instructs the corrupted parties to follow the protocol. Semi-honest adversaries model "honest but curious" behavior, where the adversary tries to learn additional information other than the output, based on the internal states of the corrupted parties.

- **Fail-stop adversaries:** a fail-stop adversary instructs the corrupted parties to follow the protocol as a semi-honest adversary, but it may also instruct a corrupted party to halt early (only sending some of its messages in a round).

- **Malicious adversaries:** a malicious adversary can instruct the corrupted parties to deviate from the protocol in any arbitrary way it chooses. There are no restrictions on the behavior of malicious adversaries.

Unless stated otherwise, we consider malicious adversaries who may arbitrarily deviate from the protocol specification. When considering malicious adversaries, there are certain undesirable actions that cannot be prevented. Specifically, parties may refuse to participate in the protocol, may substitute their local input (and enter with a different input) and may cease participating in the protocol before it terminates. Essentially, secure protocols limit the adversary to such behavior only.

We further assume that the adversary is *computationally bounded* and *static*. By computationally bounded we mean that the adversary is modeled by a non-uniform probabilistic polynomial-time interactive Turing machine. By static, we mean that at the beginning of the execution, the adversary is given a set $\mathcal{I}$ of corrupted parties which it controls.

Semi-malicious adversaries. As a stepping stone towards realizing the standard definition of secure multi-party computation against active adversaries, we consider a notion (c.f. see the work [AJW11] for further discussions) of a semi-malicious adversary that is stronger than the standard notion of semi-honest adversary. We present the formulation by Asharov, Jain, and Wichs [AJW11] as follows:

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 seeing 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.

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.

As discussed in the work [AJW11], the main benefit in considering semi-malicious adversaries is shown in the work [AJW11] (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. Thus, the complier does not increase round complexity.

A.2 Ideal Functionalities

In the following, we consider several ideal functionalities to capture MPC with fairness and guarantee of output delivery. For simplicity we ignore the session id when it is clear in the context. Our presentation follows the work [CL14].

A.2.1 Secure Computation with Guaranteed Output Delivery

This definition provides the strongest notion of security we consider. According to this definition, the protocol can terminate only when all parties receive their prescribed output. Recall that a malicious party can always substitute its input or refuse to participate. Therefore, the ideal functionality takes this inherent adversarial behavior into account by giving the adversary the ability to do this also in the ideal model. Since the adversary cannot abort the execution of the protocol in this model, fail-stop adversaries are equivalent to semi-honest adversaries (in particular, they cannot substitute their input). An ideal execution proceeds as follows:

Send inputs to the trusted functionality. Each honest party P_i sends its input x_i to the trusted party. Maliciously corrupted parties may send the trusted party arbitrary inputs as instructed by the adversary. Denote by x'_i the value sent by P_i. In the case of a semi-honest or fail-stop adversary, we require that $x'_i = x_i$.

Trusted functionality answers the parties. If x'_i is outside the domain for P_i or P_i sends no input, the trusted party sets x'_i to some predetermined default value. Then the trusted party computes $(y_1, \ldots, y_n) \leftarrow \mathcal{F}(x'_1, \ldots, x'_n)$ and sends y_i to party P_i for every $i \in [n]$.

Output. Honest parties always output the message received from the trusted party and the corrupted parties output nothing. The adversary outputs an arbitrary function of the initial inputs $\{x_i\}_{i\in\mathcal{I}}$ and the messages received by the corrupted parties from the trusted party $\{y_i\}_{i\in\mathcal{I}}$.

A.2.2 Secure Computation with Complete Fairness

This definition is similar to the previous one, except that the execution can terminate in two possible ways: the first is when all parties receive their prescribed output (as in the previous case); and the second is when all parties (including the corrupted parties) abort without receiving output. This is "fair" since in both cases the adversary receives no more information than the corrupted parties. In this definition, when sending the inputs to the trusted party, the adversary is allowed to send a special abort command. In this case, the trusted party sends a special abort symbol $\perp$ as the output to all parties. Without loss of generality, we assume that a malicious party always sends an input which is either in the corresponding input domain or `Abort`, since in case the trusted party receives a value outside of the domain, it can proceed as if `Abort` was sent. In this definition, fail-stop adversaries have the additional capability over semi-honest adversaries to abort the computation without anyone receiving output. An ideal execution proceeds as follows:

Send inputs to trust functionality. Each honest party P_i sends its input x_i to the trusted party. Maliciously corrupted parties may send the trusted party arbitrary inputs as instructed by the adversary. In addition, there exists a special `Abort` input. Denote by x'_i the value sent by P_i. We require that in the case of a semi-honest adversary $x'_i = x_i$ and fail-stop adversary $x'_i \in \{x_i, \texttt{Abort}\}$.

Trusted functionality answers the parties. If there exists an i such that $x'_i = \texttt{Abort}$, the trusted party sends $\perp$ to all parties. Otherwise, the trusted party computes $(y_1, \ldots, y_n) \leftarrow \mathcal{F}(x'_1, \ldots, x'_n)$ and sends y_i to party P_i for every $i \in [n]$.

Output. The same as the case in Section A.2.1.

B Shamir's Secret Sharing Scheme

In this section, we describe the idea of Shamir's secret sharing schemes. For more details, we refer the readers to [Sha79]. Let q be some prime and $\mathbb{Z}_q$ be the finite field of size q. Let $N \in \mathbb{N}$ and $t \geq N$ be a threshold. The goal of a t-out-of-N secret sharing scheme is to share an arbitrary value s into N shares $\mathsf{P}_1, \ldots, \mathsf{P}_N$, and for any subset (of parties) of size t can uniquely reconstruct the secret s.

The Shamir's secret scheme works as follows:

- To generate a random shares of s, we consider a random degree $t-1$ polynomial $p \in \mathbb{Z}_q[x]$ such that $p(0) = s$. The share to party P_i is $p(i)$ for $i \in [N]$.
- Given any set of parties P_S for $S \subseteq [N]$ such that $|S| = t$, the polynomial can be uniquely determined using the polynomial interpolation:
$$p(x) = \sum_{j\in S} \mu_j(x)p(j),$$
where $\mu_j(x)$ is the Lagrange polynomial with respect to the set S, i.e.,
$$\mu_j(x) = \prod_{j'\neq j;\ j'\in S} \frac{x - j'}{j - j'}.$$
Thus, the secret $s = p(0)$ can be obtained.

In this paper, we also consider shares of vectors. That is, the secret is a vector $\vec{s} \in \mathbb{Z}_q^n$ for some n. For this case, we denote $\vec{p}$ as the polynomial vector $\vec{p} = [p_1, p_2, \ldots, p_n]^T$ (where T denotes the transpose). To share $\vec{s}$, we simply generate a random polynomial vector such that $\vec{p}(0) = [p_1(0), p_2(0), \ldots, p_n(0)]^T = \vec{s}$. The reconstruction is the same as above.

C Assumptions and Other Definitions

C.1 Complexity Assumptions

We state the definition of the Learning with Error (LWE) assumption.

Definition C.1 ([Reg05]) *Let $n = n(\kappa), q = q(\kappa)$ be integers and $\chi = \chi(\kappa)$ be some distribution. Then for any polynomial $m(\kappa)$, the distribution $(\mathbf{A}, \mathbf{A} \cdot \vec{s} + \vec{e})$ is computationally indistinguishable from uniform, where $\mathbf{A}$ is a uniformly random matrix in $\mathbb{Z}_q^{m \times n}$, $\vec{s}$ is a uniformly random vector in $\mathbb{Z}_q^n$ and $\vec{e} \leftarrow \chi^m$.*

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. Then we state two useful lemmas from the work [AJL+12]. The first shows that given sufficiently many random LWE samples, the secret $\vec{s}$ is uniquely determined. The second shows that adding large noise "smudges out" any small values.

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

Lemma 3 *Let $B_1 = B_1(\kappa)$, $B_2 = B_2(\kappa)$ be positive integers and let e_1 be an integer such that $|e_1| < B_1$. Let $e_2 \leftarrow [-B_2, B_2]$ be chosen uniformly at random. Then the distribution of e_2 is statistically close to that of $e_1 + e_2$ as long as $B_1/B_2 = \mathsf{negl}(\kappa)$.*

C.2 Program Obfuscation

Definition C.2 (Circuit Obfuscator [BGI+01]) *A probabilistic algorithm O is a (circuit) obfuscator for the collection $\mathcal{C}$ of circuits if the following holds:*

- *(functionality) For every circuit $C \in \mathcal{F}$, the string $O(C)$ describes a circuit that computes the same function as C.*
- *(polynomial slowdown) There is a polynomial p such that for every circuit $C \in \mathcal{C}$, we have $|O(C)| \leq p(|C|)$.*
- *("virtual black box" (VBB) property) For any PPT $\mathcal{A}$, there is a PPT $\mathcal{S}$ and a negligible function ν such that for all circuits $C \in \mathcal{C}$, it holds that*

$$\left| \Pr\left[A(O(C)) = 1\right] - \Pr\left[\mathcal{S}^C(1^{|C|}) = 1\right] \right| \leq \nu(|C|).$$

We say that O is efficient if it runs in polynomial time. If we omit specifying the collection $\mathcal{F}$, then it is assumed to be the collection of all circuits.

D Additional Results

Recall that Garg et al. [GGHR14] were the first to show how to leverage indistinguishability obfuscation to obtain a two-round MPC protocol (security with abort). In this paper, we further weaken their assumption, and show that witness encryption (WE) is sufficient to obtain two-round MPC secure against malicious adversaries (security with abort). As we discussed in the introduction, we can achieve fairness by adding one more round. It is understood that indistinguishability obfuscation implies witness encryption [BH15] (achieving the adaptive soundness); but the other way around is not clear.

D.1 Preliminary: Witness Encryption

First, we formally define witness encryption in the same manner as in Garg et al. [GGSW13], and we define adaptive soundness in the way that Bellare and Hoang suggest [BH15].

Definition D.1 (Witness Encryption) *A witness encryption scheme for an NP language L_R (where the witness relation is R) consists of the following polynomial-time algorithms:*

- $\mathsf{Enc}_{\mathsf{WE}}(1^\kappa, \mathsf{stmt}, M)$*: encrypts a message M under an NP statement* stmt*.*
- $\mathsf{Dec}_{\mathsf{WE}}(\mathsf{ct}, w)$*: given a witness w such that $R(\mathsf{stmt}, w) = 1$, decrypt the ciphertext* ct *encrypted for the statement* stmt*.*

These algorithms should satisfy correctness and soundness as formally defined below:

Correctness. For any security parameter $\kappa \in \mathbb{N}$, for any $M \in \{0,1\}^*$, and for any $\mathsf{stmt} \in L$ such that $R(\mathsf{stmt}, w)$ holds, we have that

$$\Pr[\mathsf{Dec}_{\mathsf{WE}}(\mathsf{Enc}_{\mathsf{WE}}(1^\kappa, \mathsf{stmt}, M), w) = M] = 1$$

where probability is taken over random coins consumed by the encryption and decryption algorithms.

Adaptive soundness. We define adaptive soundness through the following game [BH15].

- $(\mathsf{stmt}, m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(1^\kappa)$.
- $b \leftarrow \{0,1\}$; $c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{WE}}(1^\kappa, \mathsf{stmt}, m_b)$
- $b' \leftarrow \mathcal{A}(\mathsf{st}, c)$
- We say that the adversary $\mathcal{A}$ wins the above game if $b' = b$ and $x \notin L$.

A witness encryption scheme is said to be adaptively secure for language L, if for every PPT adversary $\mathcal{A}$, its probability of winning the above game is bounded by $\frac{1}{2} + \mathsf{negl}(\kappa)$.

Intuitively, adaptive soundness of witness encryption guarantees that if the statement is false, then the ciphertext does not reveal anything about the encrypted message in a computational sense.

D.2 Preliminary: Two-Round MPC from Indistinguishability Obfuscation

Our central idea is to construct an indistinguishability obfuscator for *dynamic point functions* by leveraging witness encryption in a blackbox manner. We can then directly plug in this indistinguishability obfuscator into the 2-round MPC protocol by Garg et al [GGHR14].

To understand why we need an indistinguishability obfuscator for dynamic point functions, we quickly review the protocol by Garg et al. [GGHR14] below. Their construction is a compiler that starts out with a semi-honest MPC protocol π, and produces a 2-round MPC protocol secure against malicious adversaries in the UC-model.

Their prototol proceeds in two rounds:

- **Round 1:** Parties commit to their inputs and randomeness, where the commitments are generated using a CCA-secure encryption scheme. The committed randomness will be used for coin-flipping to obtain unbiased random coins. In particular, every party P_i generates and commits to randomness $\{r_{i,j}\}_{j \in [n]}$. Party P_i's randomness in the underlying protocol π is determined by $s_i := \oplus_j r_{j,i}$.
- **Round 2:**
  - Each party P_i reveals the random values $\{r_{i,j}\}_{j \neq i}$, and generates proofs that these are indeeded the random values committed to in Round 1.

– Assume that the underlying protocol π is a t-round protocol, where each party broadcasts one message per round. Each party P_i generates t obfuscations of its next-message function $(\mathsf{iO}_{i,1}, \ldots, \mathsf{iO}_{i,t})$. In more detail, each $\mathsf{iO}_{i,k}$ is an obfuscation of the following function $F_{i,k}$ that takes as input the $r_{i,j}$ values sent by all parties along with proofs that they are well-formed, and all the π-messages that were broadcast up to round $k-1$, along with the NIZK proof of correct generation of these messages. The output of the function $F_{i,k}$ is the next message of P_i in protocol π, along with a NIZK proof that it was generated correctly.

Observe that in the above, each function $F_{i,k}$ to be obfuscated is a dynamic point function. In particular, after the first-round commitment phase, the next messages of all parties in the underlying protocol π is uniquely determined – assuming that the commitment scheme is perfectly binding. We can think of each $F_{i,k}$ as being parameterized by all parties' messages sent in the first round. The function $F_{i,k}$ takes two inputs: 1) all parties' next messages in protocol π up to round $k-1$, denoted M_{k-1}; and 2) proofs attesting to the correctness of all parties' next messages so far w.r.t. the commitments sent in the first round, denoted Ω_{k-1}. (One can think of Ω_{k-1} as the collection of R, Φ, and Γ terms by Garg et al.'s notation.)

In other words, in our dynamic point function (see formal definition in Section D.3), the NP relation R is parameterized by all parties' messages sent in the first round. The unique valid statement x corresponds to the next messages of all parties up to round $k-1$. The witness w collects all the proofs terms which may not be unique.

Therefore, it suffices to show how to construct iO for dynamic point functions from witness encryption. We will show how to achieve this below. However, before that, we first formalize the notion of dynamic point function – which was informally described by Garg et al. [GGHR14].

D.3 Definitions: Dynamic Point Function and its Obfuscation

Let R denote an NP relation. If there exists *exactly one* valid statement x for which there exists some w such that $R(x, w) = 1$, then R defines a dynamic function family $\mathcal{C}_R$. A dynamic point function $C_f \in \mathcal{C}_R$ is defined as follows:

$$C_f(x, w) = \begin{cases} f(x) & \text{if } R(x, w) = 1 \\ \perp & \text{otherwise} \end{cases}$$

Clearly, a dynamic point function has only one meaningful point that does not evaluate to $\perp$.

Definition D.2 (Obfuscation of dynamic point function) *A polynomial-time algorithm $\mathcal{O}$ is said to be an obfuscator for a dynamic point function family $\mathcal{C}_R$ if the following holds:*

- *(Correctness). For all security parameters $\kappa \in \mathbb{N}$, for all $C_f \in \mathcal{C}_R$, for all input (x, w), we have that*

$$\Pr[C'(x, w) = C_f(x, w) : C' \leftarrow \mathcal{O}(1^\kappa, C_f)] = 1$$

- *(Security). There is a PPT simulator $\mathcal{S}$, such that for any PPT distinguisher $\mathcal{D}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for any $C_f \in \mathcal{C}_R$,*

$$\Pr[\mathcal{D}(1^\kappa, \mathcal{O}(1^\kappa, C_f)) = 1] - \Pr[\mathcal{D}(1^\kappa, \mathcal{S}(1^\kappa, x^*, f(x^*)) = 1] \leq \mathsf{negl}(\kappa)$$

  such that x^* is the only valid point where $R(x^*, w) = 1$ for some w.

It is not hard to show that the above definition for dynamic point function obfuscation implies iO for dynamic point functions. Specifically, for all PPT distinguisher $\mathcal{D}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for any $C_{f_0}, C_{f_1} \in \mathcal{C}_R$ which are functionally equivalent (i.e., agree on all inputs), let $y^* = f_0(x^*) = f_1(x^*)$, then both $\mathcal{O}(1^\kappa, C_{f_0})$ and $\mathcal{O}(1^\kappa, C_{f_1})$ are indistinguishable from $\mathcal{S}(1^\kappa, x^*, y^*)$.

D.4 Construction: Dynamic Point Function Obfuscation from WE

To obfuscate a dynamic point function $C_f \in \mathcal{C}_R$,

- Let $\ell := |x|$ denote the bit-length of the input x. Compute

$$(\Gamma, \{L_i^0, L_i^1\}_{i\in[\ell]}) \leftarrow \mathsf{Gb.Garble}(1^\kappa, U(f,\cdot))$$

where U is the universal circuit such that $U(f,x) := f(x)$. We assume that the garbled inputs corresponding to f are hardwired in Γ.

- For $i \in [\ell]$, $b \in \{0,1\}$ compute $c_{i,b} := \mathsf{Enc}_{\mathsf{WE}}(1^\kappa, \mathsf{stmt}, L_i^b)$ where stmt is the following statement:

$$\exists (x,w), \text{ s.t.} (R(x,w)=1) \wedge (x_i = b)$$

The obfuscation for $F_{f,R}$ is defined as the following tuple:

$$\left(\Gamma,\ \{c_{i,b}\}_{i\in[\ell], b\in\{0,1\}}\right)$$

Evaluation is done in the most natural manner. Suppose the evaluator knows some witness w such that $R(x,w)=1$. The evaluator first uses the w to decrypt the corresponding labels $\{L_i^{x_i^*}\}_{i\in[\ell]}$ where x^* is the only statement such that $R(x^*,w)=1$ for some w. Then, the evaluator uses these labels as inputs to the garbled circuit Γ, and evaluates the outcome.

Theorem D.3 *Assuming that there exists a correct and adaptively secure witness encryption scheme, then the above construction is a secure obfuscator for a dynamic point function family $\mathcal{C}_R$ by Definition D.2.*

Proof: We prove security via the following hybrid arguments.

Hybrid 0: Real world.

Hybrid 1: In the witness encryption step, replace the labels $L_i^{1-x_i^*}$ with random labels before encrypting. Hybrid 0 and Hybrid 1 are computationally indistinguishable due to the definition of dynamic point function and the security of the witness encryption scheme.

Hybrid 2: Call the garbled circuit simulator $(\Gamma^*, \{L_i\}_{i\in[\ell])}) \leftarrow \mathsf{Gb.SimGarble}(1^\kappa, f(x^*))$. Let $L_i^{x_i^*} := L_i$ for the WE encryption step. Hybrid 2 is indistinguishable from Hybrid 1 due to the security of garbled circuit. ■

Corollary D.1 (2-round MPC from witness encryption.) *Assuming that there exists a correct and adaptively secure witness encryption scheme, then there exists a 2-round MPC that is UC secure against malicious adversaries that corrupts arbitrary number of parties, in the CRS model (security with abort).*

Proof sketch. It is not hard to see that if the commitment scheme is perfectly binding, then the functions to be obfuscated $F_{i,k}$ in Garg et al.'s protocol are indeed dynamic point functions (as also observed by Garg et al. [GGHR14]), where the relation R is parameterized by all parties' messages in the first round. The rest of the proof follows directly from Garg et al. [GGHR14].

E Proof of Theorem 4.2

In this section, we describe the full analysis of the protocol in Section 4.3.

E.1 Correctness

Let $x_1, \ldots, x_N$ be arbitrary inputs of the N parties. To argue correctness, we are going to show that with overwhelming probability, the protocol outputs $y = f(x_1', \ldots, x_N')$ where x_i' is set to be either x_i or the default value. We argue this in two steps: (1) we first argue that the TFHE scheme will produce the correct output when there are no error terms (as in the second round of the protocol). (2) Then we argue that any small error terms will not affect the output. (Note that the error terms are used to prove security only).

Let $x_1, \dots, x_N$ be arbitrary inputs of the N parties. Let $S \subset [N]$ be an arbitrary non-empty set, and f be an arbitrary boolean function of depth L (we assume that the circuit only uses NAND gates and has L levels). We want to argue that for any public parameter pp and public keys $\vec{b}_1, \dots, \vec{b}_N$, the ciphertext C^* encrypts the message $f(x_1, \dots, x_N)$ with respect to the public keys $\mathsf{pk}_S = \{\mathsf{pk}_i : i \in S\}$, where C^* is obtained from the following procedure:

- Generate flexible ciphertexts $C_i = (\mathbf{W}_{i,1}, \dots, \mathbf{W}_{i,N}) \leftarrow \mathsf{TFHE.Enc}(x_i)$ for $i \in [N]$.
- Apply the transform algorithm to get $C'_i = \sum_{j\in S} \mathbf{W}_{i,j} = \mathsf{TFHE.TransCT}(C_i; S)$.
- Apply the evaluation algorithm and obtain $C^* = \mathsf{TFHE.Eval}(f, C'_1, \dots, C'_N; S)$.

By looking into the transformed ciphertexts, we can write

$$C'_i = \sum_{j\in S} \mathbf{W}_{i,j} = \mathsf{Flatten}\left(x_i \cdot I_D + \mathsf{BitDecomp}\left(\mathbf{R}_i \cdot \sum_{j\in S} \vec{b}_j \mid\mid \mathbf{R}_i \cdot \mathbf{B} \right)\right).$$

This is of exactly the same form of the GSW FHE ciphertexts [GSW13] under "the public key" $(\mathbf{B}, \sum_{j\in S} \vec{b}_j)$. Thus, by a direct calculation as the same analysis as the work [GSW13], we know that C^* is an encryption of y under the public key $(\mathbf{B}, \sum_{j\in S} \vec{b}_j)$. In particular, we have

$$\mathsf{BitDecomp}^{-1}(C^*_{\beta,1}) - \left\langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \sum_{j\in S} \vec{s}_j \right\rangle = 2^j \cdot y + \tilde{e},$$

where $|\tilde{e}| < (D+1)^L \cdot N \cdot B_\chi < q/8$. Note that the initial noise is bounded by $N \cdot B_\chi$, and one level of NAND will blow up the noise by $(D+1)$. By inspection of the protocol $\mathsf{TFHE.Dec}_S(C^*; \vec{0})$, each party P_i outputs $w_i = \langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{j\in S} \vec{p}_j(i)\rangle$. Actually these $\{w_i\}_{i\in S}$ form shares of the value $\langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{j\in S} \vec{p}_j(0)\rangle = \langle \mathsf{BitDecomp}^{-1}(C_{\beta,2}), \sum_{j\in S} \vec{s}_j\rangle$. Thus, by the end the parties will output

$$\left\lceil \frac{\mathsf{BitDecomp}^{-1}(C^*_{\beta,1}) - \langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \sum_{j\in S} \vec{s}_j \rangle}{2^\beta} \right\rfloor = y,$$

since $\tilde{e}$ is rounded. Further, if $\vec{v}$ forms shares of small error terms, then $\mathsf{TFHE.Dec}_S(C^*; \vec{v}) = \mathsf{TFHE.Dec}_S(C^*; \vec{0})$ since $\vec{v}$ will also be rounded. This completes the analysis of the correctness.

E.2 Security

To prove security, we need to argue that for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$, $\mathbf{Real}_{\pi,\mathcal{A},\mathcal{Z}} \approx \mathbf{Ideal}_{\mathcal{F}_f,\mathcal{S},\mathcal{Z}}$. We construct the simulator as follows. Let $\mathcal{A}$ be a static semi-malicious adversary who corrupts parties with indices in $\mathcal{I}$ and has auxiliary information z. Let $G = [N] \setminus \mathcal{I}$ denote the set of honest parties. (Recall that a semi-malicious adversary is required to write to his *witness tape* some randomness to explain the behavior of a message he sends in each round).

The simulator. The simulator simulates the $\mathsf{pp} = (n, q, \chi, m, B_{\mathsf{smug}}, \mathbf{B})$ faithfully as in the real protocol. Then the simulator simulates the non-corrupted parties as follows.

(First round.) The simulator simulates the first round of the $\mathsf{TFHE.Gen}(\mathsf{pp})$ protocol for the non-corrupted parties. For every honest party P_i $(i \in G)$, the simulator outputs a uniformly random vector as its output of the public key $\vec{u}_i \in \mathbb{Z}_q^m$ (instead of $\vec{b}_i = \mathbf{B} \cdot \vec{s}_i + \vec{e}_i$ as the prescribed procedure).

(Second round.) The simulator proceeds to simulate the second round of the messages that contain (1) encrypted inputs, (2) the shares of error terms, and (3) the shares of the secret keys as in the second round of $\mathsf{TFHE.Gen}$. For each honest party P_i $(i \in G)$, the simulator simulates each item as as follows:

- it simulates the ciphertexts by $\mathsf{TFHE.Enc}_i(0)$ (for every input bit).

- it simulates the error term and computes random secret shares $(r_i(1), \dots, r_i(N))$ as in the real world. Then P_i sends $r_i(j)$ to party P_j for $j \in [N]$.
- it simulates the shares of the secret keys of the honest parties by sending random values to the adversary. Let $\{\vec{\gamma}_{i,j}\}_{j \in \mathcal{I}}$ denote random values (for shares), and P_i sends $\vec{\gamma}_{i,j}$ to P_j for $j \in \mathcal{I}$. Since $|\mathcal{I}| < N/2$, the distribution of shares sent to the adversary by the simulator is identical to that in the real world. (Recall that we assume that there are secure point-to-point channel available.)

At the conclusion of the second round, the simulator reads the witness tape of the semi-malicious adversary and learns the inputs x_i and secret keys $\vec{s}_i$ and the polynomial of the error terms r_i of P_i where $i \in \mathcal{I}$ and P_i has not aborted currently. Denote all currently non-aborting parties as the set S_2, and the simulator sets all x_j's to the default value if $j \notin S_2$. Then the simulator submits the inputs to the ideal functionality and learns the output y.

(Third Round.) Given the output, the simulator is going to "reconstruct" the honest parties' messages. The simulator is going to generate messages that are consistent with the output y and previous messages. We describe the strategy as follows:

- The simulator computes C^* the same way as in the real world. In particular, he first applies the $\mathsf{TFHE.TransCT}$ algorithms on the flexible ciphertexts with respect to the set S_2, and then applies the $\mathsf{TFHE.Eval}$ computation of the residual function f^{S_2}.
- Then he picks the β-th row $C^*_\beta = (C^*_{\beta,1}, C^*_{\beta,2})$ as in the real decryption. Then he computes:

$$W = \mathsf{BitDecomp}^{-1}(C^*_{\beta,1}) - \sum_{j \in S_2 \cap \mathcal{I}} \langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \vec{s}_j \rangle - 2^\beta \cdot y.$$

  Recall that the secret keys $\vec{s}_j$'s for parties $j \in S_2 \cap \mathcal{I}$ were read from the witness tape in the previous step. So the simulator has all necessary information to compute W.

- Then the simulator proceeds as follows:
  - For $j \in \mathcal{I}$, set $\vec{\rho}_j = \sum_{i \in G} \vec{\gamma}_{i,j}$. Then let $V \subseteq G$ be an arbitrary set of size $[T/2] - |\mathcal{I}|$. Then the simulator samples random values and assigns them to $\vec{\rho}_t$ for $t \in V$.
  - Then the simulator sets $\alpha_j = \langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \vec{\rho}_j \rangle$ for $j \in \mathcal{I} \cup V$, and $\alpha_0 = W$. Then he computes other α_t's for $t \in [N] \setminus (\{0\} \cup V \cup \mathcal{I})$:

  $$\alpha_t := \sum_{j \in \{0\} \cup V \cup \mathcal{I}} \mu_j(t) \alpha_j,$$

    where $\mu_j(t)$ is the Lagrange polynomial evaluated at t. That is,

  $$\mu_j(t) = \prod_{j' \neq j;\ j' \in \{0\} \cup V \cup \mathcal{I}} \frac{t - j'}{j - j'}.$$

  Intuitively, each $\vec{\rho}_j$ for $j \in \mathcal{I} \cup V$ corresponds to the polynomial vector $\vec{p}_G := \sum_{i \in G} \vec{p}_i$ evaluated at j, and $W = \langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \vec{p}_G(0) \rangle$, where $\vec{p}_i$ is the polynomial vector followed by the shares of $\vec{s}_i$. In the simulation, the simulator does not generate $\vec{s}_i$ and then generates a polynomial and its shares. Instead, the simulator generates random shares to the adversary first (in round 2), and then figures out $W = \alpha_0$ as above. After computing α_0, α_j for $j \in \mathcal{I} \cup V$, the simulator uses the interpolation method to compute the other α_t's as above. These α_t's will be used as in the final step below.

- Finally, the simulator now simulates the messages of the honest parties. For $i \in G$, the simulator computes $\vec{z}_i = \sum_{j \in S_2 \cap \mathcal{I}} \vec{p}_j(i)$. Then the party P_i broadcasts the value

$$\langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \vec{z}_i \rangle + \alpha_i + \sum_{j \in S_2} r_j(i).$$

Now, we want to show that given an environment $\mathcal{Z}$ cannot distinguish the real world from the idea world, i.e. $\mathbf{Real}_{\mathcal{A},\pi,\mathcal{Z}} \approx \mathbf{Ideal}_{\mathcal{S},\mathcal{F}_f,\mathcal{Z}}$. This can be done by a hybrid argument. Given any $\mathcal{Z}$, first we define the following hybrid experiments ($\mathcal{Z}$ included implicitly):

Hybrid H_1: We modify the real experiment as follows. Assume (as a mental experiment) that the honest parties are given all the secret keys $\{\vec{s}_i\}_{i\in\mathcal{I}}$, the error polynomials $\{r_i\}_{i\in\mathcal{I}}$ (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. The experiment follows the real experiment in the first two rounds, and follows the ideal world in the third round.

That is, the experiment does not use the secret keys of the honest parties' to run the threshold decrypt. Instead, it first calculates the W, and then calculates $\{\alpha_t : t \in G\}$, where G is the set of honest parties. (Recall that they are supposed to be $\langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), p_G(0)\rangle$, and $\langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), p_G(i)\rangle$ for $i \in G$). As demonstrated above, these values can be obtained only from $\{\vec{s}_i\}_{i\in\mathcal{I}}$, the output y, and C^*.

Hybrid H_2: This experiment is the same as H_1 except that the honest parties send a uniformly random vector $\vec{u}_h$ as its public key.

Note that the only difference between H_2 and the ideal experiment is that in H_2, the honest parties send $\mathsf{TFHE.Enc}_i(x_i)$ in the second round, but in the ideal world, they send $\mathsf{TFHE.Enc}_i(0)$.

Then we will prove the following claims:

Claim E.1 *$\mathbf{Real}_{\mathcal{A},\pi,\mathcal{Z}} \approx H_1$.*

Proof: First we assume that the "witnesses" that $\mathcal{A}$ writes on the special witness tape on behalf of a corrupted parties P_j for $j \in \mathcal{I}$ in various rounds contains the same *consistent* values for $\vec{s}_j$ in each round. This follows by the uniqueness of secret-key (Lemma 2), which implies that the values $\vec{b}_j$'s sent on behalf of P_j information theoretically commit the adversary $\mathcal{A}$ to unique witnesses $\vec{s}_j$'s. Therefore, we will speak of well-defined, consistent values of $\vec{s}_j$'s independent of which round we are talking about.

Let G denote the set of honest parties. First we observe that

$$\Delta\left(\mathbf{Real}_{\mathcal{A},\pi,\mathcal{Z}}, H_1\right) \leq \Delta\left(X, Y\right),$$

where $\Delta(\cdot,\cdot)$ denotes the statistical difference, and X, Y are defined as follows:

$$X = \left\{\left\langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \sum_{j\in S_2} p_j(i)\right\rangle + \sum_{j\in S_2} r_j(i)\right\}_{i\in G},$$

and

$$Y = \left\{\left\langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \sum_{j\in S_2\cap\mathcal{I}} p_j(i)\right\rangle + \alpha_i + \sum_{j\in S_2} r_j(i)\right\}_{i\in G}.$$

This is because, the only difference between the real experiment and H_1 is the third-round messages, and X, Y denote the random variables according to the messages in the real and H_1 respectively. Since the polynomials $\{p_j : j \in S_2 \cap \mathcal{I}\}$ are generated identically in both experiments, we have $\Delta(X,Y) = \Delta(X', Y')$ where

$$X' = \left\{\left\langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \sum_{j\in G} p_j(i)\right\rangle + \sum_{j\in S_2} r_j(i)\right\}_{i\in G}, \text{ and } Y' = \left\{\alpha_i + \sum_{j\in S_2} r_j(i)\right\}_{i\in G}.$$

Note that $G = S_2 \setminus \mathcal{I}$. We observe that the distribution X' is identical to random secret shares (to the set G) of the value

$$\left\langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \sum_{j\in G} \vec{p}_j(0)\right\rangle + \sum_{j\in S_2} r_j(0) = \left\langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \sum_{j\in G} \vec{s}_j\right\rangle + \sum_{j\in S_2} r_j(0),$$

and Y' is identical to random secret shares (to the set G) of the value $W + \sum_{j\in S_2} r_j(0)$.

By the correctness of the fully homomorphic encryption scheme (as shown in [GSW13]), we have

$$C^*_{\beta,1} - \left\langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \sum_{j\in S_2} \vec{s}_j \right\rangle = 2^\beta \cdot y + \tilde{e},$$

for some small error $\tilde{e}$ (i.e. $|\tilde{e}| < (D+1)^L \cdot N \cdot B_\chi$). By rearranging the equation, we have:

$$\begin{aligned}\langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \sum_{j\in G} \vec{s}_j \rangle &= \mathsf{BitDecomp}^{-1}(C^*_{\beta,1}) - \sum_{j\in S_2\cap\mathcal{I}} \langle \mathsf{BitDecomp}^{-1}(C^*_{\beta,2}), \vec{s}_j \rangle - 2^\beta \cdot y + \tilde{e} \\ &= W + \tilde{e}.\end{aligned}$$

Since $|\tilde{e}|/B_{\mathsf{smug}} = \mathsf{negl}(\kappa)$ by the setting of the parameters, by Lemma 3 we know $\Delta(W+\tilde{e}+\sum_{j\in S_2} r_j(0), W+\sum_{j\in S_2} r_j(0)) = \mathsf{negl}(\kappa)$. As we have argued that X' and Y' are random shares of these two values (distributions), we can conclude that $\Delta(X', Y') = \mathsf{negl}(\kappa)$. This completes the proof of the claim. ■

Claim E.2 $H_1 \approx H_2$.

This follows directly from the LWE assumption.

Claim E.3 $H_2 \approx \boldsymbol{Ideal}_{\mathcal{S},\mathcal{F}_f,\mathcal{Z}}$.

We note that the honest parties of the two the experiments do not use the secret keys $\vec{s}_j$'s for $j \in G$ to decrypt. Thus, the claim follows from Lemma 1 that shows the scheme is semantically secure.

The proof of the theorem follows immediately from these claims.