Secure Massively Parallel Computation for Dishonest Majority

Rex Fernando* Ilan Komargodski† Yanyi Liu‡ Elaine Shi§

July 9, 2021

Abstract

This work concerns secure protocols in the massively parallel computation (MPC) model, which is one of the most widely-accepted models for capturing the challenges of writing protocols for the types of parallel computing clusters which have become commonplace today (MapReduce, Hadoop, Spark, etc.). Recently, the work of Chan et al. (ITCS '20) initiated this study, giving a way to compile any MPC protocol into a secure one in the common random string model, achieving the standard secure multi-party computation definition of security with up to 1/3 of the parties being corrupt.

We are interested in achieving security for much more than 1/3 corruptions. To that end, we give two compilers for MPC protocols, which assume a simple public-key infrastructure, and achieve semi-honest security for all-but-one corruptions. Our first compiler assumes hardness of the learning-with-errors (LWE) problem, and works for any MPC protocol with "short" output—that is, where the output of the protocol can fit into the storage space of one machine, for instance protocols that output a trained machine learning model. Our second compiler works for any MPC protocol (even ones with a long output, such as sorting) but assumes, in addition to LWE, indistinguishability obfuscation and a circular secure variant of threshold FHE. Both protocols allow the attacker to choose corrupted parties based on the trusted setup, an improvement over Chan et al., whose protocol requires that the CRS is chosen independently of the attacker's choices.

1 Introduction

In the past two decades, the model of a sequential algorithm executing on a RAM machine with one processor has become increasingly impractical for large-scale datasets. Indeed, numerous programming paradigms, such as MapReduce, Hadoop, and Spark, have been developed to utilize parallel computation power in order to manipulate and analyze the vast amount of data that is available today. Starting with the work of Karloff, Suri, and Vassilvitskii [49], there have been several attempts at formalizing a theoretical model capturing such frameworks [3, 34, 47, 49, 52, 54, 57, 70]. Today the most widely accepted model is called the *Massively Parallel Computation* (MPC) model. Throughout this paper, whenever the acronym MPC is used, it means "Massively Parallel Computation" and not "Multi-Party Computation".

The MPC model is believed to best capture large clusters of Random Access Machines (RAM), each with a somewhat considerable amount of local memory and processing power, yet not enough to store the massive amount of available data. Such clusters are operated by large companies such as Google or Facebook. To be more concrete, letting N denote the total number of data records, each machine can only store $s = N^{\epsilon}$ records locally for some $\epsilon \in (0, 1)$, and the total number of machines is $m \approx N^{1-\epsilon}$ so that they can jointly store the entire data-set. One should think of N as huge, say tens or hundreds of petabytes, and ϵ as small, say 0.2.[1] In many MPC algorithms it is also okay if $m \cdot s = N \cdot \log^c N$ for some constant $c \in \mathbb{N}$ or even $m \cdot s = N^{1+\theta}$ for some small constant $\theta \in (0, 1)$, but not much larger than that (see, e.g., [49, 54, 1, 4]).

*UCLA and NTT Research. Email: rex1fernando@gmail.com.

†NTT Research and Hebrew University. Email: ilan.komargodski@ntt-research.com.

‡Cornell University. Email yl2866@cornell.edu.

§Cornell University. Email: runting@gmail.com.

[1] If N is one Petabyte (10^6 Gigabytes), then the storage of each machine in the cluster needs to be < 16 Gigabytes.

The primary metric for the complexity of algorithms in this model is their *round complexity*. Computations that are performed within a machine are essentially "for free". The rule of thumb in this context is that algorithms that require $o(\log_2 N)$ rounds (e.g., $O(1)$ or $O(\log\log N)$)) are considered *efficient*. With the goal of designing efficient algorithms in the MPC model, there is an immensely rich algorithmic literature suggesting various non-trivial efficient algorithms for tasks of interest, including graph problems [1, 3, 4, 5, 7, 8, 9, 10, 12, 17, 18, 16], [31, 34, 41, 43, 38, 53, 54, 62, 68], clustering [13, 15, 36, 42, 73] and submodular function optimization [67, 37, 52, 58].

Secure MPC. In a very recent work, Chan, Chung, Lin, and Shi [27] initiated the study of secure computation in the MPC model. Chan et al. [27] showed that any task that can be efficiently computed in this model can also be securely computed with comparable efficiency. More precisely, they show that any MPC algorithm can be compiled to a secure counterpart that defends against a malicious adversary who controls up to $1/3 - \eta$ fraction of machines (for an arbitrarily small constant η), where the security guarantee is similar to the one in *cryptographic secure multiparty computation*. In other words, an adversary is prevented from learning anything about the honest parties' inputs except for what the output of the functionality reveals. The cost of this compilation is very small: the compiled protocol only increases the round complexity by a constant factor, and the space required by each machine only increases by a multiplicative factor that is a fixed polynomial in the security parameter. Since round complexity is so important in the MPC setting, it is crucial that these cost blowups are small. Indeed, any useful compiler must preserve even a sublogarithmic round complexity. The security of their construction relies on the Learning With Errors (LWE) assumption and they further rely on the existence of a common random string that is chosen *after* the adversary commits to its corrupted set.

Why is secure MPC hard? Since there is a long line of work studying secure multiparty computation (starting with [45, 19]), a natural first question is whether these classical results extend to the MPC model in a straightforward way. The crucial aspect of algorithms in the MPC model which makes this task non-trivial is the combination of the space constraint with the required small round complexity. Indeed, many existing techniques from the standard secure computation literature fail to extend to this model, since they either require too many rounds or they require each party to store too much data. For instance, it is impossible for any one party to store commitments or shares of all other parties' inputs, a common requirement in many secure computation protocols (e.g., [50, 65]). This also rules out naively adapting protocols that rely on more modern tools such as threshold FHE [6, 59, 35], as they also involve a similar first step. Even previous work that focused on large-scale secure computation [22] required one broadcast message *per party*, which either incurs a large space overhead or a large blowup in the number of rounds. Chan et al. [27] give an exciting feasibility result for secure protocols in this model, but their construction, as mentioned, has some significant limitations: (1) it only tolerates at most ≈1/3 corruptions, and (2) it relies on a trusted setup which must be chosen *after* the choice of the corrupted parties. Whether these limitations are inherent in this new model remains an intriguing open question.

This work. We consider the setting of *all-but-one corruptions*, where the computation is performed in the MPC model but security is required even for a single honest machine if all other players are controlled by an adversary. In the classical secure multi-party computation literature this setting is referred to as the *dishonest majority* setting and generic protocols tolerating such adversarial behaviour are well known (e.g., [45]). In contrast, in the MPC model, it is a-priori not even clear that such a generic result can be obtained with the space and round complexity constraints. This raises the following question, which is the focus of this work:

Is there a generic way to efficiently compile any massively parallel protocol into a secure version that tolerates all-but-one corruptions?

1.1 Our Results

We answer the above question in the affirmative. We give two compilers that can be used to efficiently compile any algorithm in the MPC model into an algorithm that implements the same functionality also in the MPC model, but now secure even in the presence of an attacker who controls up to $m-1$ of the m machines. Both of our protocols handle *semi-honest* attackers who are assumed to follow the specification of the protocol.

In terms of trusted setup, in both of our protocols we assume that there is a public-key infrastructure (PKI) which consists of a $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_m)$: a single public key and m secret keys, one per machine. Machine $i \in [m]$ knows pk and sk_i, whose size is independent of N (and none of the other secret keys). Crucially, our protocols allow the adversary to choose the corrupted parties based on the setup phase, an improvement over the construction of [27], for which there is an obvious and devastating attack if the adversary can choose corrupted parties based on the common random string.

Notation and parameters. Let N denote the bit size of the data-set[2] and suppose that each machine has space $s = N^{\epsilon}$ for some fixed constant $\epsilon \in (0,1)$. We further assume that the number of machines, m, is about $N^{1-\epsilon}$ or even a little bigger. The security parameter is denoted λ and it is assumed that $N < \lambda^c$ for some $c \in \mathbb{N}$ and $s > \lambda$.

1.1.1 Secure MPC with Short Outputs

Our first result is a compiler that fits best for tasks whose output is "short". By short we mean that it fits into the memory of (say) a single machine. The compiler blows up the number of rounds by a constant and the space by a fixed polynomial in the security parameter, which is identical to the efficiency of the compiler in [27]. For security, we rely on the LWE assumption [69].

While at first it may seem that this compiler is quite restricted in the class of algorithms it supports, in fact, there are many important and central functionalities that fit in this class. For instance, this class contains all graph problems whose output is somewhat succinct (like finding a shortest path in a graph, a minimum spanning tree, a small enough connected component, etc). Even more impressively, all submodular maximization problems, a class of problems that captures a wide variety of problems in machine learning, fit into this class [67].

Theorem 1 (Secure MPC for Short Output, Informal)**.** *Assume hardness of LWE. Given any massively parallel computation (MPC) protocol* Π *which after* R *rounds results in an output of size* $\leq s$ *for party 1 and no output for any other party, there is a secure MPC algorithm* $\tilde{\Pi}$ *that securely realizes* Π *with semi-honest security in the presence of an adversary that statically corrupts up to* $m-1$ *parties. Moreover,* $\tilde{\Pi}$ *completes in* $O(R)$ *rounds, consumes at most* $O(s) \cdot \mathsf{poly}(\lambda)$ *space per machine, and incurs* $O(m \cdot s) \cdot \mathsf{poly}(\lambda)$ *total communication per round.*

As mentioned above, by security we mean an analogue of standard cryptographic multiparty computation security, adapted to the massively parallel computation (MPC) model. We use the LWE assumption to instantiate a secure variant of an n-out-of-n threshold fully-homomorphic scheme (FHE) [6, 20] which supports "incremental decoding". This is an alternative to the standard decoding procedure of threshold FHE schemes which is suited to work in the MPC model. See Section 2 for details.

We prove that our construction satisfies semi-honest security where the attacker gets to choose its corrupted set *before* the protocol begins but *after* the public key is published. (In comparison, recall that [27] had their attacker commit on its corrupted set before even seeing the CRS.)

1.1.2 Secure MPC with Long Outputs

Our second result is a compiler that works for *any* protocol in the MPC model. Many MPC protocols perform tasks whose output is much larger than what fits into one machine. Such tasks may include, for example, the task of sorting the input. Here the result of the protocol is that each machine contains a small piece of the output, which is considered to be the concatenation of all machines' outputs in order. Our second compiler can be used for such functionalities.

In this construction we rely, in addition to LWE, on a circular secure variant of the threshold FHE scheme from the short output protocol and also on indistinguishability obfuscation [14, 39, 71]. The compiler achieves the same round and space blowup as the short-output compiler.

[2] We assume for simplicity that a data record takes up one bit.

Theorem 2 (Secure MPC for Long Output, Informal)**.** *Assume the existence of an circular secure n-out-of-n threshold FHE scheme with incremental decoding, along with iO and hardness of LWE. Given any massively parallel computation (MPC) protocol* Π *that completes in* R *rounds, there is a secure MPC algorithm* $\tilde{\Pi}$ *that securely realizes* Π *with semi-honest security in the presence of an adversary that statically corrupts up to* $m-1$ *parties. Moreover,* $\tilde{\Pi}$ *completes in* $O(R)$ *rounds, consumes at most* $O(s) \cdot \mathsf{poly}(\lambda)$ *space per machine, and incurs* $O(m \cdot s) \cdot \mathsf{poly}(\lambda)$ *total communication per round.*

1.2 Related Work

The cryptography literature has extensively studied secure computation on parallel architectures, but most existing works focus on the PRAM model (where each processing unit has $O(1)$ local storage) [61, 22, 23, 30, 56, 32, 33, 29, 28, 2]. Most real-world large-scale parallel computation is now done on large clusters which are much more accurately modeled by the MPC architecture, and the aforementioned works usually do not apply to this setting. Other distributed models of computations have been considered in cryptographic contexts. Parter and Yogev [63, 64] considered secure computation on graphs in the so-called CONGEST model of computation (where each message is of size at most $O(\log N)$ bits). Boyle et al. [25] proposed an approach to compile a fixed-communication protocol to a secure counterpart that preserves the bottleneck complexity (i.e., the maximum communication complexity required by any party) of the original protocol. They achieve malicious security by adopting a strong SNARK where extraction has additive polynomial overhead. Their approach can be used to obtain a maliciously secure Massively Parallel Computation protocol for short outputs; however, it is not applicable to long outputs.

Paper Organization

An overview of our constructions is given next in Section 2. Some standard preliminaries and the building blocks that we use in our construction are formally defined in Section 3. The MPC model is formally defined in Section 4. The compiler for short output protocols appears in Section 5 and the compiler for long output protocols is in Section 6.

2 Technical Overview

In this section we give the high-level overview of our protocols. Let us briefly recall the model. The total input size contains N bits and there are about $m \approx N^{1-\epsilon}$ machines, each having space $s = N^{\epsilon}$. In every round, each machine can send and receive at most s bits since its local space is bounded (e.g., a machine cannot broadcast a message to everyone in one round). We are given some protocol in the MPC model that computes some functionality $f\colon (\{0,1\}^{l_{in}})^m \to (\{0,1\}^s)^m$, where $l_{in} \leq s$, and we would like to compile it into a secure version that computes the same functionality. We would like to preserve the round complexity up to constant blowup, and to preserve the space complexity as much as possible. Moreover, we want semi-honest security, which means there must exist a simulator which, without the honest parties' inputs, can simulate the view of a set of corrupted parties, provided the parties do not deviate from the specification of the protocol.

Since our goal is to use cryptographic assumptions to achieve security for MPC protocols, we introduce an additional parameter λ, which is a security parameter. One should assume that N is upper bounded by some large polynomial in λ and that s is large enough to store $O(\lambda)$ bits.

We first note that we can start by assuming that the communication patterns, i.e., the number of messages sent by each party, the size of messages, and the recipients, do not leak anything about the parties' inputs. We call a protocol that achieves this *communication oblivious.* A generic transformation for any MPC protocol was shown by [27], which achieved communication obliviousness with constant blowup in rounds and space.

2.1 The Short Output Protocol

We start with a protocol in the easier case where the underlying MPC results with a "short" output, meaning that it fits into the memory of a single machine (say the first one).

In a nutshell, the idea is as follows: we want to execute an encrypted version of the (insecure) MPC algorithm using a homomorphic encryption scheme. In the classical setting of secure computation this idea was extensively used in threshold/multi-key FHE based solutions, for instance, in [6, 55, 60, 26, 66, 20, 11] There, in a high-level, each party first broadcasts an encryption of its input. Then each party can (locally) homomorphically compute the desired function over the combined inputs of all parties, and finally all parties participate in a joint decryption protocol that allows them to decrypt the output. Moreover, this joint decryption protocol does not allow any party to decrypt any ciphertext beyond the output ciphertext. The classical joint decryption protocol is completely non-interactive: each party broadcasts a "partial decryption" value so that each party who holds partial decryptions from all other parties can locally decode the final output of the protocol.

Recall that in our setting each party has bounded space and so it is impossible for any party to store all partial decryptions and so the joint decryption protocol described above cannot work in the MPC model. To get around this, we relax the joint decryption protocol by allowing it to be *interactive*. To this end, we design a new joint decryption protocol that splits the process of "combining" partial decryption into many rounds (concretely, $\log_\lambda m \in O(1)$ rounds). We use the additive-secret-sharing threshold FHE scheme of Boneh et al. [20] and modify their decryption procedure, as we explain next.

At a high-level, the ciphertext in the simplest variant of Boneh et al.'s [20] scheme is a GSW [40] ciphertext, and each party's secret key is a linear share of the GSW secret key. In addition, partial decryption works in the same way as the GSW decryption using the secret key share with an additional re-randomization, and then the final decryption phase is just a linear function that combines the partial decryptions and a final step of rounding. [3] We observe that the first part of final decryption, which is just a linear function, can be executed in a tree-like fashion, so that if each party has a partial decryption, no party will need to store more than a few partial decryptions at a time.

Our trick is to adjust the parameters of this tree to be aligned with the MPC model. We let each machine hold about λ different partial decryptions (causing a λ blow up in space) which causes the depth of the tree to be roughly $\log_\lambda m$. Since m is bounded by some fixed polynomial in λ this is still $O(1)$. Overall, this step adds $O(1)$ rounds of communication and results with a single party knowing the output. As a small technical note, to simulate the view of set of corrupted parties which do not learn the output, we require one additional property of the threshold FHE scheme: it must be possible to simulate partial decryptions of an incomplete set $I' \subsetneq [m]$ of parties without knowing even the output of the circuit. This requirement is not captured in the original definition of threshold FHE in [20], but we show that their construction satisfies it.

2.2 The Long Output Protocol

Here, we would like to support MPC protocols whose output is "long", namely, each party will have an output. Directly extending the short output protocol fails. Indeed, there, we used a tree-like protocol to gradually "aggregate" the sum of all partial decryption at a single machines's memory. In the current case, each party needs to do the same procedure to recover its own output. Since we have a bound on the total communication of each party, we cannot run all gradual decryptions in parallel, so this requires about m/ϵ rounds (which is way too much).

Recall that the goal of the decryption phase is for the parties to learn the decryption of its output, without learning the decryptions of any other ciphertext. If we can somehow construct a decryption phase where the communication is independent of the output size, we would have a valid long output protocol. This is non-trivial: what we essentially need is some "limited" master secret key, which somehow only decrypts a limited set of ciphertexts, and nothing else. Moreover, we need to be able to generate this key within the limitations of the MPC model: no single machine can even hold the complete set of ciphertext which this secret key is supposed to decrypt.

Let us define the functionality of this "limited" master secret key more formally. It will be convenient to describe it as a circuit. Ideally, the circuit has hardwired the secret keys from all parties along with all ciphertexts which correspond to the output of the computation. Each party would be able to submit its output to the circuit, and the circuit would be able to check if this ciphertext is a member of the valid set, and decrypt if this is the case. Even ignoring security (namely, that a machine can learn all keys; we will address this later), there are two efficiency problems: first, the circuit contains m ciphertexts, and the second is that it contains m secret keys (and recall that $m \gg s$).

[3] Note that although the Shamir-based TFHE scheme in [20] requires a field size which is polynomial in the number of parties n, the field size in the simpler additive-based scheme is independent of n, which is crucial in our construction.

To solve the first problem, instead of storing the ciphertexts explicitly, we use a succinct commitment thereof. We need a way for the parties to collectively compute this commitment in the MPC model and without increasing the number of rounds too much. To this end, we use a variant of Merkle commitments with larger arity. We note that the tree structure of Merkle commitments suits our model very well: if a single machine is responsible for computing the label of a single node in the tree, we achieve a low-communication-complexity protocol relatively easily. Then, if we set the arity to be $O(\lambda)$, the number of rounds will be roughly $\log_\lambda m$, which is constant assuming m is at most a fixed polynomial in λ.

To solve the second problem, we observe an important property about the basic n-out-of-n threshold FHE scheme of Boneh et al. Namely, in this scheme, the public key is a GSW public key, each party's secret key is a linear share of the corresponding GSW secret key, and encryption under the threshold FHE scheme is simply a GSW encryption with this public key. This means that knowing the sum of all parties' secret keys is sufficient to decrypt, and this sum is compact.

So we have a feasible circuit with the functionality we need in order to implement a "limited" master secret key. We of course need a secure version of this circuit, which will not leak the master secret key hardcoded in the circuit. To do this we use indistinguishability obfuscation. We give a high-level overview of the techniques which we use in conjunction with obfuscation to achieve security. Since we want to be able to simulate the view of the corrupted parties, we need a simulated version of the circuit, which has no master secret key embedded but which can still produce the decrypted outputs. The main idea for how we overcome this is to exploit the fact that the simulator is allowed to set the randomness of the corrupted parties. We will use the Merkle commitment to force each party to input their randomness to the circuit, and when simulating we will embed the output in this randomness, padded with a PRF. The circuit can then unpad and use this as its output without knowing the secret key. This is a somewhat standard technique in iO literature first used by [46]. One technical detail is that since iO only guarantees indistinguishability against circuits which are functionally equivalent, we need a succinct commitment which can guarantee statistical binding for some indices. This type of primitive, a somewhere-statistically-binding (SSB) hash, was also constructed by [46] from the learning-with-errors (LWE) assumption. We observe that the construction of [46] also uses a tree structure similar to a Merkle tree, which allows the machines to collectively compute the commitment without too much communication or storage.

Now that we have a way to generate a "limited" secret key, the question is, how can the parties do this without leaking their secret key shares? We need to somehow assemble this obfuscated circuit, which has the master secret key embedded, even though no party is allowed to know the master secret key. Our key idea is to leverage our short-output secure MPC protocol for this purpose: we can use that protocol to securely compute the obfuscated circuit! The short-output protocol guarantees that parties learn nothing about the execution of the protocol beyond the output and their inputs, and this is exactly what we need in order to compute the obfuscated circuit without revealing the master secret key.

One final technical challenge we need to overcome is that an SSB hash commitment does not guarantee privacy; it may leak information about the committed values. In order to achieve output privacy, we introduce an extra step in the protocol where each party pads their encrypted output before committing. We refer to Section 6 for details.

On the necessity of a PKI. Our constructions require a public-key infrastructure (PKI); a trusted party must generate a (single) public key and (many) secret key shares which it distributes to each machine. We do not know if this is necessary, but at least we argue that known techniques from the classical secure computation literature do not work in the MPC model (and so drastically new ideas are needed). Indeed, classically, secure multi-party computation protocols avoid using a PKI by using threshold multi-key FHE (e.g., [6, 55, 60, 26, 66, 11]), where each party generates its own key pair and uses the concatenation of all public keys as the master public key. This does not extend to our setting, since the number of machines is much larger than the space of each individual machine (and so a machines cannot even store all public keys). Of course, obtaining our results without a PKI is a natural open problem.

3 Preliminaries

For $x \in \{0,1\}^*$, let $x[a:b]$ be the substring of x starting at a and ending at b. A function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}$ is *negligible* if it is asymptotically smaller than any inverse-polynomial function, namely, for every constant $c > 0$ there exists an integer N_c such that $\mathsf{negl}(\lambda) \leq \lambda^{-c}$ for all $\lambda > N_c$. Two sequences of random variables $X = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ are *computationally indistinguishable* if for any non-uniform PPT algorithm $\mathcal{A}$ there exists a negligible function negl such that $\left|\Pr[\mathcal{A}(1^\lambda, X_\lambda) = 1] - \Pr[\mathcal{A}(1^\lambda, Y_\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$ for all $\lambda \in \mathbb{N}$.

3.1 Threshold FHE with Incremental Decryption

We will use a threshold FHE scheme with an "incremental" decryption procedure, specialized for the MPC model. Our definition follows that of [48].

An n-out-of-n threshold fully homomorphic encryption scheme with incremental decryption is a tuple $(\mathsf{TFHE.Setup}, \mathsf{TFHE.Enc}, \mathsf{TFHE.Eval}, \mathsf{TFHE.Dec}, \mathsf{TFHE.PartDec}, \mathsf{TFHE.CombineParts}, \mathsf{TFHE.Round})$ of algorithms which satisfy the following properties:

- $\mathsf{TFHE.Setup}(1^\lambda, n) \to (pk, sk_1, \ldots, sk_n)$: On input the security parameter λ and the number of parties n, the setup algorithm outputs a public key and a set of secret key shares.
- $\mathsf{TFHE.Enc}_{pk}(m) \to ct$: On input a public key pk and a plaintext $m \in \{0,1\}^*$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{TFHE.Eval}(C, ct_1, \ldots, ct_k) \to \hat{ct}$: On input a public key pk, a circuit $C : \{0,1\}^{l_1} \times \cdots \times \{0,1\}^{l_k} \to \{0,1\}^{l_o}$, and a set of ciphertexts $ct_1, \ldots, ct_k$, the evaluation algorithm outputs a ciphertext $\hat{ct}$.
- $\mathsf{TFHE.Dec}_{sk}(ct) \to m$: On input the master secret key $sk_1 + \cdots + sk_n$ and a ciphertext ct, the decryption algorithm outputs the plaintext m.
- $\mathsf{TFHE.PartDec}_{sk_i}(ct) \to p_i$: a ciphertext ct and a secret key share sk_i, the partial decryption algorithm outputs a partial decryption p_i for party P_i.
- $\mathsf{TFHE.CombineParts}(p_I, p_J) \to p_{I \cup J}$: On input two partial decryptions p_I and p_J, the combine algorithm outputs another partial decryption algorithm $p_{I \cup J}$
- $\mathsf{TFHE.Round}(p) \to m$: On input a partial decryption p, the rounding algorithm outputs a plaintext m.

Compactness of ciphertexts: There exists a polynomial p such that $|ct| \leq \mathsf{poly}(\lambda) \cdot |m|$ for any ciphertext ct generated from the algorithms of the TFHE, and $p_i \leq \mathsf{poly}(\lambda) \cdot |m|$ as well for all i.[4]

Correctness with local decryption: For all $\lambda, n, C, m_1, \ldots, m_k$, the following condition holds. For $(pk, sk_1, \ldots, sk_n) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, n)$, $ct_j \leftarrow \mathsf{TFHE.Enc}_{pk}(m_j)$ for $j \in [k]$, $\hat{ct} \leftarrow \mathsf{TFHE.Eval}(C, ct_1, \ldots, ct_k)$, and $p_i \leftarrow \mathsf{TFHE.PartDec}_{sk_i}(\hat{ct})$, take any binary tree with n leaves labeled with the p_i, and with each non-leaf node v labeled with $\mathsf{TFHE.CombineParts}(p_l, p_r)$, where p_l is the label of v's left child and p_r is the label of v's right child. Let ρ be the label of the root; then

$$\Pr\left[\mathsf{TFHE.Round}(\rho) = C(m_1, \ldots, m_k)\right] = 1 - \mathsf{negl}(\lambda).$$

Correctness of MSK decryption: For all $\lambda, n, C, m_1, \ldots, m_k$, the following condition holds. For $(pk, sk_1, \ldots, sk_n) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, n)$, $ct_i \leftarrow \mathsf{TFHE.Enc}_{pk}(m_i)$ for $i \in [k]$, $\hat{ct} \leftarrow \mathsf{TFHE.Eval}(C, ct_1, \ldots, ct_k)$,

$$\Pr\left[\mathsf{TFHE.Dec}_{sk}(\hat{ct}) = C(m_1, \ldots, m_k)\right] = 1 - \mathsf{negl}(\lambda),$$

where $sk = sk_1 + \cdots + sk_n$.

[4] As noted in the technical overview, although this does not hold for the Shamir-based TFHE scheme in [20], it *does* hold for the simpler additive-based TFHE scheme given in the same paper.

Semantic (and circular) security of encryption: We give two alternative definitions of semantic security, the standard one and a notion of circular security. For any PPT adversary $\mathcal{A}$, the following experiment $\mathsf{Expt}_{\mathcal{A},\mathrm{TFHE},\mathrm{sem}}$ outputs 1 with $1/2 + \mathsf{negl}(\lambda)$ probability:

$\mathsf{Expt}_{\mathcal{A},\mathrm{TFHE},\mathrm{sem}}$:

1. The challenger runs $(pk, sk_1, \ldots, sk_n) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, n)$ and provides pk to $\mathcal{A}$.
2. $\mathcal{A}$ outputs a set $I \subsetneq [n]$ and a message m; for circular security m can contain special symbols $\llcorner sk_i \lrcorner$.
3. The challenger provides $\{sk_i\}_{i \in I}$ to $\mathcal{A}$.
4. In circular security, the challenger computes m' by replacing every symbol $\llcorner sk_i \lrcorner$ with the secret key sk_i. In normal semantic security the challenger sets $m' = m$.
5. The challenger chooses $b \overset{\$}{\leftarrow} \{0, 1\}$; if $b = 0$ then the challenger sends $\mathsf{TFHE.Enc}_{pk}(m')$, and if $b = 1$ then the challenger sends $\mathsf{TFHE.Enc}_{pk}(0^|m'|)$.
6. $\mathcal{A}$ outputs a guess b'. The experiment outputs 1 if $b = b'$.

(Circular) Simulation security: There exists a simulator ($\mathsf{TFHE.Sim.Setup}$, $\mathsf{TFHE.Sim.Query}$) such that for any PPT $\mathcal{A}$, the following experiments $\mathsf{Expt}_{\mathcal{A},\mathrm{TFHE},\mathrm{real}}$ and $\mathsf{Expt}_{\mathcal{A},\mathrm{TFHE},\mathrm{ideal}}$ are indistinguishable:

$\mathsf{Expt}_{\mathcal{A},\mathrm{TFHE},\mathrm{real}}$:

1. The challenger runs $(pk, sk_1, \ldots, sk_n) \leftarrow \mathsf{TFHE.Setup}(1^\lambda)$ and provides pk to $\mathcal{A}$.
2. $\mathcal{A}$ outputs a set $I \subsetneq [n]$ and messages $m_1, \ldots, m_k$, along with $\{r_j\}_J$ for some subset $J \subset [k]$. In addition, for circular simulation security each m_i can contain special symbols $\llcorner sk_{i'} \lrcorner$.
3. In circular simulation security, for each $i \in [k]$, the challenger computes m'_i by replacing every symbol $\llcorner sk_{i'} \lrcorner$ with the secret key $sk_{i'}$. In normal simulation security, the challenger sets $m'_i = m_i$.
4. The challenger provides $\{sk_i\}_{i \in I}$ to $\mathcal{A}$ and $\{\mathsf{TFHE.Enc}_{pk}(m'_i)\}_{i \in [k]}$ to $\mathcal{A}$. For each $i \in [k]$, if the adversary supplied randomness r_i, then this randomness is used as the randomness for encrypting m'_i.
5. $\mathcal{A}$ issues a polynomial number of adaptive queries of the form (I', C), and for each query the challenger computes $\hat{ct} \leftarrow \mathsf{TFHE.Eval}(C, ct_1, \ldots, ct_k)$ and responds with $\{\mathsf{TFHE.PartDec}_{sk_i}(\hat{ct})\}_{i \in I'}$.
6. At the end of the experiment, $\mathcal{A}$ outputs a distinguishing bit b.

$\mathsf{Expt}_{\mathcal{A},\text{TFHE},\text{ideal}}$:

1. The challenger runs $(pk, sk'_1, \ldots, sk'_n, \sigma_{sim}) \leftarrow \mathsf{TFHE.Setup}(1^\lambda)$ and provides pk to $\mathcal{A}$.
2. $\mathcal{A}$ outputs a set $I \subsetneq [n]$ and messages $m_1, \ldots, m_k$, along with $\{r_j\}_J$ for some subset $J \subset [k]$. In addition, for circular simulation security each m_i can contain special symbols $\llcorner sk_{i'} \lrcorner$.
3. In circular simulation security, for each $i \in [k]$, the challenger computes m'_i by replacing every symbol $\llcorner sk_{i'} \lrcorner$ with the secret key $sk_{i'}$. In normal simulation security, the challenger sets $m'_i = m_i$.
4. The challenger runs $(\{sk_i\}_{i \in I}, \sigma_{sim}) \leftarrow \mathsf{TFHE.Sim.Setup}(pk, I)$ and provides $\{sk_i\}_{i \in I}$ and $\{\mathsf{TFHE.Enc}_{pk}(m'_i)\}_{i \in [k]}$ to $\mathcal{A}$. For each $i \in [k]$, if the adversary supplied randomness r_i, then this randomness is used as the randomness for encrypting m'_i.
5. $\mathcal{A}$ issues a polynomial number of adaptive queries of the form (I', C), and the challenger runs the simulator $\{p_i\}_{i \in I'} \leftarrow \mathsf{TFHE.Sim.Query}(C, \{ct_i\}_{i \in [k]}, \{r_j\}_{j \in J}, C(m'_1, \ldots, m'_k), I', \sigma_{sim})$ and responds with $\{p_i\}_{i \in I'}$.
6. At the end of the experiment, $\mathcal{A}$ outputs a distinguishing bit b.

Simulation of incomplete decryptions: We additionally require that, for the above experiments, if $I \cup I' \neq [n]$, then it is possible to simulate partial decryptions without knowing the circuit output. In other words, if $I \cup I' \neq [n]$ then in the ideal world the challenger can compute

$$\{p_i\}_{i \in I'} \leftarrow \mathsf{TFHE.Sim.Query}(C, \{ct_i\}_{i \in [k]}, \{m'_j, r_j\}_{j \in J}, \perp, I', \sigma_{sim})$$

in step 4 above, and indistinguishability still holds.

Although this additional requirement is not explicit in the simulation security definition of [48], it follows implicitly from the fact that semantic security holds whenever the adversary does not have all secret keys sk_i. More specifically, assume the adversary requests an "incomplete" partial decryption set I' from the challenger, where $I \cup I' \neq [m]$. This means that for all $i \in [m] \setminus (I \cup I')$, the adversary receives no information at all about sk_i, so by TFHE semantic security it is possible to switch all encryptions for $i \notin J$ (i.e. where the adversary does not supply the encryption randomness) to 0. Thus to simulate partial decryptions for I', it is only necessary to know the output of C over the inputs $m'_i, i \in J$, and $0, i \notin J$. Since the TFHE simulator receives m'_i for all $i \in J$, it can thus simulate partial decryptions without knowing the output of C over the true inputs.

The next theorem states that a threshold FHE (TFHE) scheme with incremental decryption exists under the Learning with Errors (LWE) assumption.

Theorem 3. *Assuming LWE, there exists a threshold FHE (TFHE) scheme with incremental decryption satisfying the above requirements except for circular security.*

Proof sketch. We use the most basic construction of [20] and observe that it can be modified to satisfy the incremental decryption property as follows. In their decryption procedure, one gets all partial decryptions and they are added together and then a non-linear rounding is performed. We obtain incrementality by separating the two parts into two procedures. The first only performs the first part of adding up partial decryptions–this can be done incrementally since this is a linear operation. The second operation is the rounding operation which is executed in the end.

To see why the simulation of incomplete decryptions property holds, note that the secret keys of the parties are linear shares of a GSW secret key. This means that if $I \cup I' \neq [n]$ then the distribution of shares corresponding to $I \cup I'$ are identical to uniform. Thus the simulator can pick uniform random sk_i for each $i \in I'$ in order to simulate partial decryptions without knowing the circuit evaluation. □

Remark 1. *We note that we will use a plain threshold FHE (TFHE) scheme with incremental decryption in the protocol for short output functionalities (see Section 5) and so that one can be based on the hardness of LWE. However, the long output protocol (see Section 6) will require a circular secure version of threshold FHE (TFHE) scheme with incremental decryption (defined above) which we do not know how to base on any standard assumption, except by assuming that the construction from Theorem 3 satisfies it).*

3.2 Somewhere Statistically Binding Hash

A somewhere statistically binding (SSB) hash [46] consists of the following algorithms, which satisfy the properties below:

- $\mathsf{SSB.Setup}(1^\lambda, L, d, f, i^*) \to h$: On input integers L, d, f, and an index $i^* \in [f^d L]$, outputs a hash key h.
- $\mathsf{SSB.Start}(h, x) \to v$: On input h and a string $x \in \{0,1\}^L$, output a hash tree leaf v.
- $\mathsf{SSB.Combine}(h, \{v_i\}_{i\in[f]}) \to \hat{v}$: On input h and f hash tree nodes $\{v_i\}_{i\in[f]}$, output a parent node $\hat{v}$.
- $\mathsf{SSB.Verify}(h, i, x_i, z, \{v\}) \to b$: On input h, and index i, a string x_i, a hash tree root z, and a set $\{v\}$ of nodes, output 1 iff $\{v\}$ consists of a path from the leaf corresponding to x_i to the root z, as well as the siblings of all nodes along this path.

Correctness: For any integers L, d, and f, and any indices i^*, j, strings $\{x_i\}_{i\in[f^d]}$ where $|x_i| = L$, and any $h \leftarrow \mathsf{SSB.Setup}(1^\lambda, L, d, f, i^*)$, if $\{v\}$ consists of a path in the tree generated using $\mathsf{SSB.Start}(h, \cdot)$ and $\mathsf{SSB.Combine}(h, \cdot)$ on the leaf strings $\{x_i\}_{i\in[f^d]}$, from the leaf corresponding to x_j to the root z, along with the siblings of all nodes along this path, then $\mathsf{SSB.Verify}(h, j, x_j, z, \{v\}) = 1$.

Compactness of commitment and openings: All node labels generated by the $\mathsf{SSB.Start}$ and $\mathsf{SSB.Combine}$ algorithms are binary strings of size $\mathsf{poly}(\lambda) \cdot L$.

Index hiding: Consider the following game between an adversary $\mathcal{A}$ and a challenger:

1. $\mathcal{A}(1^\lambda)$ chooses L, d, and f, and two indices i_0^* and i_1^*.
2. The challenger chooses a bit $b \leftarrow_\$ \{0,1\}$ and sets $h \leftarrow \mathsf{SSB.Setup}(1^\lambda, L, d, f, i_b^*)$.
3. The adversary gets h and outpus a bit b'. The game outputs 1 iff $b = b'$.

We require that no PPT $\mathcal{A}$ can win the game with non-negligible probability.

Somewhere statistically binding: For all λ, L, d, and f, i^*, and for any key $h \leftarrow \mathsf{SSB.Setup}(1^\lambda, L, d, f, i^*)$, there do not exist any values z, x, x', $\{v\}$, $\{v'\}$ such that $\mathsf{SSB.Verify}(h, i^*, x, z, \{v\}) = \mathsf{SSB.Verify}(h, i^*, x', z, \{v'\}) = 1$.

Theorem 4 ([46, Theorem 3.2])**.** *Assume LWE. Then there exists an SSB hash construction satisfying the above properties.*

3.3 Indistinguishability Obfuscation for Circuits

Let $\mathcal{C}$ be a class of Boolean circuits. An obfuscation scheme for $\mathcal{C}$ consists of one algorithm iO with the following syntax.

$iO(C \in \mathcal{C}, 1^\lambda)$: The obfuscation algorithm is a PPT algorithm that takes as input a circuit $C \in \mathcal{C}$, security parameter λ. It outputs an obfuscated circuit.

An obfuscation scheme is said to be a secure indistingushability obfuscator for $\mathcal{C}$ [14, 39, 71] if it satisfies the following correctness and security properties:

- Correctness: For every security parameter λ, input length n, circuit $C \in \mathcal{C}$ that takes n bit inputs, input $x \in \{0,1\}^n$, $C'(x) = C(x)$, for $C' \leftarrow iO(C, 1^\lambda)$.
- Security: For every PPT adversary $\mathcal{A} = (A_1, A_2)$, the following experiment outputs 1 with at most $1/2 + \mathsf{negl}(\lambda)$:

  EXPERIMENT $\mathsf{Expt}_{\mathcal{A},\mathrm{iO}}$:

  1. $(C_0, C_1, \sigma) \leftarrow \mathcal{A}_1(1^\lambda)$
  2. If $|C_0| \neq |C_1|$, or if either C_0 or C_1 have different input lengths, then the experiment outputs a uniformly random bit.

     Else, let n denote the input lengths of C_0, C_1. If there exists an input $x \in \{0,1\}^n$ such that $C_0(x) \neq C_1(x)$, then the experiment outputs a uniformly random bit.
  3. $b \leftarrow \{0,1\}$, $\tilde{C} \leftarrow iO(C_b, 1^\lambda)$.
  4. $b' \leftarrow \mathcal{A}_2(\sigma, \tilde{C})$.
  5. Experiment outputs 1 if $b = b'$, else it outputs 0.

3.4 Puncturable Pseudorandom Functions

We use the definition of puncturable PRFs given in [72], given as follows. A puncturable family of PRFs F is given by a triple of turing machines $\mathsf{PPRF.KeyGen}$, $\mathsf{PPRF.Puncture}$, and F, and a pair of computable functions $n()$ and $m()$, satisfying the following conditions:

- **Functionality preserved under puncturing:** For every PPT adversary $\mathcal{A}$ such that $\mathcal{A}(1^\lambda)$ outputs a set $S \subseteq \{0,1\}^{n(\lambda)}$, then for all $x \in \{0,1\}^{n(\lambda)}$ where $x \notin S$, we have that:

$$\Pr\left[F(K,x) = F(K_S,x) \mid \begin{array}{l} K \leftarrow \mathsf{PPRF.KeyGen}(1^\lambda), \\ K_S \leftarrow \mathsf{PPRF.Puncture}(K,S) \end{array}\right] = 1$$

- **Pseudorandom at punctured points:** For every PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$ such that $\mathcal{A}_1(1^\lambda)$ outputs a set $S \subseteq \{0,1\}^{n(\lambda)}$ and state σ, consider an experiment where $K \leftarrow \mathsf{PPRF.KeyGen}(1^\lambda)$ and $K_S \leftarrow \mathsf{PPRF.Puncture}(K,S)$. Then we have

$$\left|\Pr\left[\mathcal{A}_2(\sigma, K_S, S, F(K,S)) = 1\right] - \Pr\left[\mathcal{A}_2(\sigma, K_S, S, U_{m(\lambda)|S|}) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

  where $F(K,S)$ denotes the concatenation of $F(K,x)$ for all $x \in S$ in lexicographic order and U_ℓ denotes the uniform distribution over ℓ bits.

Theorem 5 ([44, 21, 24, 51])**.** *If one-way functions exist, then for all efficiently computable $n(\lambda)$ and $m(\lambda)$ there exists a puncturable PRF family that maps $n(\lambda)$ bits to $m(\lambda)$ bits.*

4 Model

4.1 Massively Parallel Computation (MPC)

We now describe the Massively Parallel Computation (MPC) model. This description is an adaptation of the description in [27]. Let N be the input size in bits and $\epsilon \in (0,1)$ a constant. The MPC model consists of m parties, where $m \in [N^{1-\epsilon}, \mathsf{poly}(N)]$ and each party has a local space of $s = N^\epsilon$ bits. Hence, the total space of all parties is $m \cdot s \geq N$ bits. Often in the design of MPC algorithms we also want that the total space is not too much larger than N, and thus many works assume that $m \cdot s = \tilde{O}(N)$ or $m \cdot s = O(N^{1+\theta})$ for some small constant $\theta \in (0, 1)$. The m parties are pairwise connected, so every party can send messages to every other party.

Protocols in the MPC model work as follows. At the beginning of a protocol, each party receives N/m bits of input, and then the protocol proceeds in rounds. During each round, each party performs some local computation bounded by $\mathsf{poly}(s)$, and afterwards may send messages to some other parties through pairwise channels. A well-formed MPC protocol must guarantee that each party sends and receives at most s bits each round, since there is no space to store more messages. After receiving the messages for this round the party appends them to its local state. When the protocol terminates, the result of the computation is written down by all machines, i.e., by concatenating the outputs of all machines. Every machine's output is also constrained to at most s bits. An MPC algorithm may be randomized, in which case every machine has a sequential-access random tape and can read random coins from the random tape. The size of this random tape is not charged to the machine's space consumption.

Communication Obliviousness: In this paper we will assume that the underlying MPC protocol discussed is *communication-oblivious*. This means that in each round, the number of messages, the recipients, and the size of each message are determined completely independently of all parties' inputs. More formally, we assume that there is an efficient algorithm which, given an index i and round number j, outputs the set of parties P_i sends messages to in round j, along with number of bits of each message. The work of [27] showed that this is without loss of generality: any MPC protocol can be compiled into an communication-oblivious one with constant round blowup. We also assume for simplicity that the underlying MPC protocol is given in the form of a set of circuits describing the behavior of each party in each round (one can emulate a RAM program with storage s with a circuit of width $O(s)$).

4.2 Secure Massively Parallel Computation

We are interested in achieving *secure* MPC: we would like protocols where, if a subset of the parties are corrupted, these parties learn nothing from an execution of the protocol beyond their inputs and outputs. We focus on semi-honest security, where all parties follow the protocol specification completely even if they are corrupted. We will also work in the PKI model, where we assume there is a trusted party that runs a setup algorithm and distributes a public key and secret keys to each party.

For an MPC protocol Π and a set I of corrupted parties, denote with $\mathsf{view}_I^{\Pi}(\lambda, \{(x_i, r_i)\}_{i\in[m]})$ the distribution of the view of all parties in I in an execution of Π with inputs $\{(x_i, r_i)\}$. This view contains, for each party $P_i, i \in I$, P_i's secret key sk_i, inputs (x_i, r_i) to the underlying MPC protocol, the random coins it uses in executing the compiled protocol, and all messages it received from all other parties throughout the protocol. We argue the existence of simulator S, a polynomial-time algorithm which takes the public key and the set I off corrupted parties and generates a view indistinguishable from $\mathsf{view}_I^{\Pi}(\lambda, \{(x_i, r_i)\}_{i\in[m]})$.

Definition 1. *We say that an MPC protocol* Π *is* semi-honest secure in the PKI model *if there exists an efficient simulator S such that for all $\{(x_i, r_i)\}_{i\in[m]}$, and all $I \subsetneq [m]$ chosen by an efficient adversary after seeing the public key, $S(\lambda, pk, I\{(x_i, r_i)\}_{i\in I}, \{y_i\}_{i\in I})$ is computationally indistinguishable from* $\mathsf{view}_I^{\Pi}(\lambda, \{(x_i, r_i)\}_{i\in[m]})$.

Note that in this definition we allow the simulator to choose each corrupted party's secret key and the random coins it uses.

5 Secure MPC for Short Output

In this section, we prove the following theorem:

Theorem 6 (Secure MPC for Short Output)**.** *Assume hardness of LWE. Suppose that $s = N^{\epsilon}$ and that m is upper bounded by a fixed polynomial in N. Let λ denote a security parameter, and assume $\lambda \le s$ and that $N \le \lambda^c$ for some fixed constant c. Given any massively parallel computation (MPC) protocol* Π *that completes in R rounds where each of the m machines has s local space, and assuming* Π *results in an output of size $l_{out} \le s$ for party 1 and no output for any other party, there is a secure MPC algorithm* $\tilde{\Pi}$ *in the PKI setting that securely realizes* Π *with semi-honest security in the presence of an adversary that statically corrupts up* *to $m-1$ parties. Moreover, $\tilde{\Pi}$ completes in $O(R)$ rounds, consumes at most $O(s)\cdot\mathsf{poly}(\lambda)$ space per machine, and incurs $O(m\cdot s)\cdot\mathsf{poly}(\lambda)$ total communication per round.*

The rest of this section is devoted to the proof of Theorem 6

5.1 Assumptions and Notation

We assume, without loss of generality, the following about the massively parallel computation (MPC) protocol which we will compile (these assumptions are essentially the same as in the previous section):

- The protocol takes R rounds, and is represented by a family of circuits $\{M_{i,j}\}_{i\in[m],j\in[R]}$, where $M_{i,j}$ denotes the behavior of party P_i in round j. In the proof of security we will also use the circuit M, the composition of all $M_{i,j}$, which takes in all parties' initial states and outputs the combined output of the protocol.
- The protocol is communication-oblivious: during round j, each party P_i sends messages to a prescribed number of parties, each of a prescribed number of bits, and that these recipients and message lengths are efficiently computable independent of P_i's state in round j.
- $M_{i,j}$ takes as input P_i's state $\sigma_{j-1}\in\{0,1\}^{\le s}$ at the end of round $j-1$, and outputs P_i's updated state σ_j. We assume σ_j includes P_i's outgoing messages for round j, and that these messages are at a predetermined location in σ_j. Let $\mathsf{MPCMessages}(i,j)$ be an efficient algorithm which produces a set $\{(i',s_{i'},e_{i'})\}$, where $\sigma[s_{i'}:e_{i'}]$ is the message for $P_{i'}$.
- At the end of each round j, P_i appends all messages received in round j to the end of σ_j in arbitrary order.
- The parties' input lengths are all l_{in}, and the output length is l_{out}.

We assume the following about the Threshold FHE (TFHE) scheme:

- For simplicity, we assume each ciphertext ct has size blowup λ.
- If ct is a valid ciphertext for message m, then $ct[\lambda\cdot(i-1):\lambda\cdot i]$ is a valid ciphertext for the i-th bit of m.
- We assume the TFHE scheme takes an implicit depth parameter, which we set to the depth of M; we omit this in our descriptions for simplicity.

5.2 The Protocol

We now give the secure MPC protocol. The protocol proceeds in two phases: first, each party encrypts its initial state under pk, and the parties carry out an encrypted version of the original (insecure) MPC protocol using the TFHE evaluation function. Second, P_1 distributes the resulting ciphertext, which is an encryption of the output, and all parties compute and combine their partial decryptions so that P_1 learns the decrypted output. This second phase crucially relys on the fact that the TFHE scheme partial decryptions can be combined locally in a tree.

The formal description of the protocol is below. Note that we use two subprotocols $\mathsf{Distribute}$ and $\mathsf{Combine}$, which are given after the main protocol.

Short Output Protocol

Setup: Each party P_i knows a public key pk along with a secret key sk_i, where $(pk,sk_1,\dots,sk_m)\leftarrow\mathsf{TFHE.Setup}(1^\lambda,m)$.

Input: Party P_i has input x_i and randomness r_i to the underlying MPC protocol.

Encrypted MPC Phase: For the first R rounds, the behavior of each party P_i is as follows:

- **Before starting:** P_i computes $ct_{\sigma_{i,0}} \leftarrow \mathsf{TFHE.Enc}_{pk}((x_i, r_i))$, its encrypted initial state.
- **During round j:** P_i starts with a ciphertext $ct_{\sigma_{i,j-1}}$, and does the following:
    1. Compute $ct_{\sigma_{i,j}} \leftarrow \mathsf{TFHE.Eval}(M_{i,j}, ct_{\sigma_{i,j-1}})$
    2. For each $(i', s_{i'}, e_{i'}) \in \mathsf{MPCMessages}(i, j)$, send $ct_{\sigma_{i,j}}[\lambda \cdot s_{i'} : \lambda \cdot e_{i'}]$ to party $P_{i'}$.
    3. For each encrypted message ct_m received in round j, append to $ct_{\sigma_{i,j}}$.

Distributed Output Decryption Phase: At the end of the encrypted execution of the MPC protocol, P_1's resulting ciphertext $ct_{\sigma_{1,R}} = ct_o$ is an encryption of the output of the protocol, and the parties do the following:

1. **All parties:** Run $\mathsf{Distribute}(ct_o)$.
2. **Each party P_i:** Compute $ct_{o,i} \leftarrow \mathsf{TFHE.PartDec}_{sk_i}(ct_o)$.
3. **All parties:** Run $\mathsf{Combine}(\mathsf{TFHE.CombineParts}, \{c_{o,i}\}_{i\in[m]})$; P_1 obtains the resulting ρ.
4. **Output:** P_1 runs $\mathsf{TFHE.Round}(\rho)$ to obtain a decryption of the output of the underlying MPC protocol.

$\mathsf{Distribute}(x)$:

Parameters: Let the fan-in f be $s/(\lambda|x|)$. Let $t = \lceil \log_f m \rceil$.

Round k: In this round, the parents are all P_i such that $i \equiv 0 \pmod{f^{t-k}}$, and the children are all P_j such that $j \equiv 0 \pmod{f^{t-k+1}}$ but $j \not\equiv 0 \pmod{f^{t-k}}$. Each parent P_i sends x to all its child nodes.

The protocol stops after t rounds. After this point all nodes have x.

$\mathsf{Combine}(\mathsf{op}, \{x_i\}_{i\in[m]})$:

Parameters: Assume op is associative and commutative, $|x_i| = |x_j|$ for all i, j, and that $|x_i \,\mathsf{op}\, x_j| = |x_i| = |x_j|$. Let the fan-in f be $s/(\lambda|x|)$.

Start: Each node P_i sets $x_{i,0} \leftarrow x_i$.

Round k: In this round, the parents are all P_i such that $i \equiv 0 \pmod{f^k}$, and the children are all P_j such that $j \equiv 0 \pmod{f^{k-1}}$ but $j \not\equiv 0 \pmod{f^k}$. Each child P_j sends $x_{j,k-1}$ to its parent P_i. P_i sets $x_{i,k} \leftarrow x_{j_s,k-1} \,\mathsf{op}\, x_{j_s+1,k-1} \,\mathsf{op} \ldots \mathsf{op}\, x_{j_e,k-1}$, where j_s is the index of the first child of P_i, and j_e is the index of the last child.

End: After $t = \lceil \log_f m \rceil$ rounds, P_1 has $x_{1,t} = x_1 \,\mathsf{op} \ldots \mathsf{op}\, x_m$.

5.3 Correctness and Efficiency

Correctness follows directly from correctness of the TFHE scheme. We proceed with the efficiency analysis. The encrypted MPC phase takes exactly R rounds. The distributed output decryption phase consists of $\mathsf{Distribute}$ and $\mathsf{Combine}$, both of which take $\lceil \log_f m \rceil$ rounds, where f is the fan-in parameter, defined to be $s/(\lambda l_{out})$. This means that the total number of rounds taken by $\mathsf{Distribute}$ and $\mathsf{Combine}$ is $2\frac{\log m}{\log(s\lambda/l_{out})}$ rounds, so the total round complexity is $R + 2\frac{\log m}{\log(s\lambda/l_{out})}$. Recall that $m = N^{c'}$ where c' is constant, so

$$\begin{aligned} \frac{\log m}{\log(s\lambda/l_{out})} &= \frac{\log \lambda^{c \cdot c'}}{\log(s\lambda/l_{out})} \\ &\leq \frac{\log \lambda^{c \cdot c'}}{\log \lambda} \leq c \cdot c'. \end{aligned}$$

Each party P_i in the encrypted MPC phase uses $s \cdot \mathsf{poly}(\lambda)$ space. During the $\mathsf{Distribute}$ and $\mathsf{Combine}$ subprotocols, P_i also uses $s \cdot \mathsf{poly}(\lambda)$, since it stores $s\lambda/l_{out}$ ciphertexts, each of size $l_{out}\lambda$. So the total space complexity is also $s \cdot \mathsf{poly}(\lambda)$.

5.4 Security

To prove security, we exhibit a semi-honest simulator for the protocol given above. This simulator will generate a view of an arbitrary set of corrupted parties using only the corrupted parties' inputs and randomness and the output of the protocol, which will be indistinguishable from the view of the corrupted parties in an honest execution of the protocol. Note that the simulator receives the public key which is assumed to be generated honestly by the TFHE setup algorithm, and also receives the set I as input. This allows the corrupted set I to be chosen based on the public key.

The behavior of the simulator is described below.

Short Output Simulator

Input: The simulator receives the corrupted set I, the public key pk, the corrupted parties' inputs and randomness $\{(x_i, r_i)\}_{i \in I}$, and, if $1 \in I$, the output y.

Simulated Setup: To generate the corrupted parties' secret keys, the simulator uses the TFHE simulated setup: $(\{sk_{c,i}\}_{i \in I}, \sigma_{sim}) \leftarrow \mathsf{TFHE.Sim.Setup}(pk, I)$.

After initializing the PKI, the simulator carries out a virtual execution of the protocol to generate the corrupted parties' views.

Simulated Encrypted MPC Phase: For the first R rounds, the behavior of the simulator is as follows:

- **Before starting:**
  - **For each corrupted party P_i:** The simulator generates uniform randomness r_i and then encrypts P_i's inputs and randomness under the public key: $ct_{\sigma_{i,0}} \leftarrow \mathsf{TFHE.Enc}_{pk}((x_i, r_i))$.
  - **For each honest party $P_{i'}$:** The simulator computes an encryption of 0: $ct_{\sigma_{i,0}} \leftarrow \mathsf{TFHE.Enc}_{pk}((0^{|x_i|}, 0^{|r_i|}))$
- **During round j:** The simulator carries out round j in the same way as in the real world.

Distributed Output Decryption Phase: At the end of the encrypted execution of the MPC protocol, the simulator has P_1's resulting ciphertext $ct_{\sigma_{1,R}} = ct_o$. It then does the following:

1. **On behalf of all parties:** Run $\mathsf{Distribute}(ct_o)$.
2. **For each corrupted P_i:** Compute $ct_{o,i} \leftarrow \mathsf{TFHE.PartDec}_{sk_i}(ct_o)$.
3. Invoke the TFHE simulator to obtain simulated partial decryptions: $\{ct_{o,i'}\}_{i \notin I} \leftarrow \mathsf{TFHE.Sim.Query}(M, \{ct_{\sigma_{i,0}}\}_{i\in[m]}, y, [m] \setminus I, \sigma_{sim})$, or if $1 \notin I$, $\{ct_{o,i'}\}_{i \notin I} \leftarrow \mathsf{TFHE.Sim.Query}(M, \{ct_{\sigma_{i,0}}\}_{i\in[m]}, \perp, [m] \setminus I, \sigma_{sim})$.
4. Compute $ct_{o,i} \leftarrow \mathsf{TFHE.PartDec}_{sk_i}(ct_o)$.
5. **On behalf of all parties:** Run $\mathsf{Combine}(\mathsf{TFHE.CombineParts}, \{c_{o,i}\}_{i\in[m]})$; P_1 obtains the resulting ρ.
6. **Output:** P_1 runs $\mathsf{TFHE.Round}(\rho)$ to obtain a decryption of the output of the underlying MPC protocol.

We now prove indistinguishability of the simulated views in the short output protocol from the real-world views. That is, we show that no efficient adversary which corrupts an arbitrary set of parties can distinguish between the output of the simulator and the view of the corrupted parties in an honest execution of the protocol. We show this via a sequence of computationally indistinguishable hybrids, where in the first hybrid the output of the simulator corresponds to the real world, and in the last hybrid the simulator's output corresponds to the ideal world.

- **Hybrid H_0:** In this hybrid, the simulator behaves identically to the real world, setting the corrupted parties' secret keys to be the ones generated by the TFHE setup, and running the real-world protocol.
- **Hybrid H_1:** In this hybrid, the simulator behaves the same as in H_0, except it generates the corrupted parties' secret keys via $\mathsf{TFHE.Sim.Setup}$, and uses $\mathsf{TFHE.Sim.Query}$ to generate the honest parties' partial decryptions for the output. Note that if $1 \notin I$ then the simulator passes $\perp$ to $\mathsf{TFHE.Sim.Query}$ instead of the output $M(x_1, \ldots, x_m)$.
- **Hybrid H_2:** In this hybrid, the simulator behaves the same as in H_1, except that it sets the encryptions of each honest party's input and randomness to 0: $ct_{\sigma_{i,0}} \leftarrow \mathsf{TFHE.Enc}_{pk}((0^{|x_i|}, 0^{|r_i|}))$. H_2 is identical to the ideal world.

We now show indistinguishability of each successive pair of hybrids, relying on the properties of the TFHE scheme which are defined in Section 3.1.

Claim 1. *Assuming simulation security of the TFHE scheme, and simulation of incomplete decryptions, the output of the simulator in H_0 is indistinguishable from the output of the simulator in H_1.*

Proof. Assume that there is an efficient adversary $\mathcal{A}$ which distinguishes between H_0 and H_1. We use $\mathcal{A}$ to build a reduction $\mathcal{A}'$ against simulation security of the TFHE scheme.

$\mathcal{A}'$ performs the same steps as the simulator in H_0, except that it interacts with the challenger to obtain the public key, the corrupted parties' secret keys, the initial ciphertexts for every party, and the partial decryptions of the honest parties. It first receives pk from the challenger, then sends this to $\mathcal{A}$, receives I from $\mathcal{A}$, along with the plaintexts $\{(x_i, r_i)\}_{i\in[m]}$, where x_i and r_i are P_i's input and randomness for the underlying MPC protocol, and sends this to the challenger. When it receives back ciphertexts $\{ct_i\}_{i\in[m]}$, it then performs the encrypted MPC phase in the same way as in H_0, and then finally, for the distributed output computation phase, it sends the query (I', M), where M is the circuit representing the underlying MPC protocol and I' is the set of honest parties whose partial decryptions are seen by parties in I. Note that if $I' \cup I \neq [m]$, which happens when $1 \notin I$, then the challenger uses $\perp$ instead of $M(x_1, \ldots, x_m)$ when evaluating $\mathsf{TFHE.Sim.Query}$. When $\mathcal{A}'$ receives the partial decryptions of the honest parties, it then uses these partial decryptions to perform the rest of the procedure in the same way as H_0. Once it has finished, it sends the views of the corrupted parties, and sends this view to $\mathcal{A}$. It outputs the output of $\mathcal{A}$.

If the TFHE simulation security challenger enacts the real-world experiment, then the view of $\mathcal{A}$ is identical to H_0. If the TFHE challenger enacts the ideal-world experiment then the view of $\mathcal{A}$ is identical to H_1. Thus if $\mathcal{A}$ distinguishes between the hybrids then $\mathcal{A}'$ is an efficient distinguisher between the real and ideal experiments of the TFHE simulation game. □

Claim 2. *Assuming semantic security of the TFHE scheme, the output of the simulator in H_1 is indistinguishable from the output of the simulator in H_2.*

Proof. Assume that there is an efficient adversary $\mathcal{A}$ which distinguishes between H_1 and H_2 with non-negligible probability. We use $\mathcal{A}$ to build a reduction $\mathcal{A}'$ against semantic security of the TFHE scheme.

$\mathcal{A}'$ sends the set I of corrupted parties to the TFHE challenger. It receives pk and the secret keys sk_i for $i \in I$. It then performs the same steps as the simulator in H_1, except that it sends the honest parties' inputs and randomness to the challenger, and uses the ciphertexts which the challenger provides as the honest parties' encrypted initial state. It sends the view of the corrupted parties to $\mathcal{A}$, and outputs the output of $\mathcal{A}$.

If the TFHE semantic security challenger sends ciphertexts with the true values, then the view of $\mathcal{A}$ is identical to H_1. If the TFHE challenger sends encryptions of 0 then the view of $\mathcal{A}$ is identical to H_2. Thus, if $\mathcal{A}$ can distinguish between the hybrids then $\mathcal{A}'$ is a successful efficient adversary against the TFHE semantic security game. □

On the source of randomness. The massively parallel computation model states that a party should not incur a space penalty for the random coins it uses. For simplicity, we did not address this part of the model in our construction, but a simple modification allows our protocol to support arbitrarily many random coins. We can do this by having the randomness embedded in the circuit $M_{i,j}$ for each step of the underlying MPC protocol, and having each party rerandomize the ciphertexts encrypting the MPC messages before sending, the standard technique for circuit privacy in FHE, to hide this randomness.

6 Long Output

We now discuss our long-output result. The theorem we prove is below.

Theorem 7 (Secure MPC for Long Output)**.** *Assume the existence of an n-out-of-n threshold FHE scheme with circular security, along with iO and LWE. Suppose that $s = N^{\epsilon}$ and that m is upper bounded by a fixed polynomial in N. Let λ denote a security parameter, and assume $\lambda \leq s$ and that $N \leq \lambda^c$ for some fixed constant c. Given any massively parallel computation (MPC) protocol Π that completes in R rounds where each of the m machines has s local space, and assuming Π results in each party having an output of size $l_{out} \leq s$, there is a secure MPC algorithm $\tilde{\Pi}$ that securely realizes Π with semi-honest security in the presence of an adversary that statically corrupts up to $m-1$ parties. Moreover, $\tilde{\Pi}$ completes in $O(R)$ rounds, consumes at most $O(s) \cdot \mathsf{poly}(\lambda)$ space per machine, and incurs $O(m \cdot s) \cdot \mathsf{poly}(\lambda)$ total communication per round.*

The rest of this section is devoted to proving Theorem 7.

6.1 Assumptions and Notation

We assume, without loss of generality, the following about the massively parallel computation (MPC) protocol which we will compile:

- The protocol takes R rounds, and is represented by a family of circuits $\{M_{i,j}\}_{i\in[m],j\in[R]}$, where $M_{i,j}$ denotes the behavior of party P_i in round j. In the proof of security we will also use the circuit M, the composition of all $M_{i,j}$, which takes in all parties' initial states and outputs the combined output of the protocol.
- The protocol is oblivious: during round j, each party P_i sends messages to a prescribed number of parties, each of a prescribed number of bits, and that these recipients and message lengths are efficiently computable independent of P_i's state in round j.

- $M_{i,j}$ takes as input P_i's state $\sigma_{j-1} \in \{0,1\}^{\leq s}$ at the end of round $j-1$, and outputs P_i's updated state σ_j. We assume σ_j includes P_i's messages for round j, and that these messages are at a predetermined location in σ_j. Let $\mathsf{MPCMessages}(i,j)$ be an efficient algorithm which produces a set $\{(i', s_{i'}, e_{i'})\}$, where $\sigma[s_{i'} : e_{i'}]$ is the message for $P_{i'}$.
- At the end of each round j, P_i appends all messages received in round j to the end of σ_j in arbitrary order.
- Each party's input is of size l_{in} and its output is of size l_{out}.

We assume the following about the TFHE scheme:

- For simplicity, we assume each ciphertext ct has size blowup λ.
- If ct is a valid ciphertext for message m, then we assume $ct[\lambda \cdot (i-1) : \lambda \cdot i]$ is a valid ciphertext for the i-th bit of m.
- We assume the TFHE scheme takes an implicit depth parameter, which we set to the maximum depth of M, $\mathsf{SSBDistSetup}$, or $\mathsf{GenerateCircuit}$; we omit this in our descriptions for simplicity.

6.2 The Protocol

We now give the secure MPC protocol. Recall that we are working under a PKI, so every party P_i knows the public key along with its secret key sk_i. At a high level, the protocol is divided into two main phases, as in the previous protocol, with the major differences occurring in the second phase. In the first phase, as in the short-output protocol, each party encrypts its initial state under pk, and the parties carry out an encrypted version of the original (insecure) MPC protocol using the TFHE evaluation function. In the second phase, the parties interact with each other so that all parties obtain an obfuscation of a circuit which will allow them to decrypt their outputs and nothing else. This involves carrying out a subprotocol CalcSSBHash in which the parties collectively compute a somewhere-statistically-binding (SSB) commitment to their ciphertexts along with some randomness. Recall that an SSB hash is a construction Merkle-tree which is designed specifically to enable security proofs when using iO.

We briefly explain CalcSSBHash. The purpose of this protocol is for all parties to know an SSB commitment z to their collective inputs, and for each party P_i to know an opening π_i for its respective input. We will perform this process over a tree with arity f (which we will specify later), mirroring the Merkle-like tree of the SSB hash. In the first round, the parties use SSB.Start, and then send the resulting label to the parties $P_{i'}$, $i' \equiv 0 \pmod{f}$ (call these nodes the parents). Each of these parties $P_{i'}$ then uses SSB.Combine on the labels $\{y_{i,0}\}$ of its children to get a new combined label $y_{i',1}$, and then all the $P_{i'}$ parties send their new labels to $P_{i''}$, $i'' \equiv 0 \pmod{f^2}$. In addition, since the string each party P_i' now has a part of its children's openings, namely $y_{i',1}$ and the set $\{y_{i,0}\}$ of sibling labels, it sends $\pi_{i,1} = (y_{i',1}, \{y_{i,0}\})$ to each of its children.

This process completes within $2\lceil \log_f m \rceil$ rounds, where in each round the current layer calculates new labels and sends them to the new layer of parents, and each layer sends any $\pi_{i,j}$ received from its parent to all its children. At the end, all parties will know z and π_i.

The formal description of the protocol is below. Note that we use the subprotocols Distribute and CalcSSBHash; Distribute was defined in the previous section, and CalcSSBHash is defined after the main protocol.

Long Output Protocol

Setup: Each party P_i knows a public key pk along with a secret key sk_i, where $(pk, sk_1, \ldots, sk_m) \leftarrow \mathsf{TFHE.Setup}(1^\lambda, m)$.

Input: Each party P_i has input x_i and randomness r_i to the underlying MPC protocol.

Encrypted MPC Phase: For the first R rounds, the behavior of each party P_i is exactly as in the encrypted MPC phase of the short output protocol.

Output Padding Phase: Assume without loss of generality each party's plaintext output in the underlying MPC protocol is of size L. After the R rounds of the encrypted MPC protocol are done, each party P_i does the following:

1. Compute a random string $pad_i \in \{0,1\}^{l_{out}}$.
2. Calculate $ct_{pad_i} \leftarrow \mathsf{TFHE.Enc}_{pk}(pad_i)$.
3. Calculate $ct_{o,i} \leftarrow \mathsf{TFHE.Eval}(\oplus, ct_{pad_i}, ct_{\sigma_{i,R}})$, the TFHE evaluation of the circuit which pads $\sigma_{i,R}$ with pad_i.

Output Circuit Generation Phase: At the end of the previous phase, each party P_i has an encryption $ct_{o,i}$ of their output padded with pad_i. The parties then coordinate with each other in a manner which is now described, so that at the end P_1 has an obfuscation of the circuit $C_{sk,z}$, defined below.

1. Each party P_i chooses a uniform random string $r_{h,i}$.
2. All parties run the short-output compiler from the previous section over the protocol $\mathsf{SSBDistSetup}(2l_{out}, 1, \{r_{h,i}\})$ defined below, so that P_1 obtains an SSB hash key h.
3. The parties run the protocol $\mathsf{Distribute}(h)$.
4. Each party chooses a uniform random string $r_{o,i}$ of size l_{out}, and the parties run the protocol $\mathsf{CalcSSBHash}(h, \{(ct_{o,i}, r_{o,i})\})$ defined below, so that each party P_i obtains an SSB commitment z and an opening π_i to $(ct_{o,i}, r_{o,i})$.
5. Each party chooses a uniform random string $r_{iO,i}$, and the parties run the short-output compiler over the protocol $\mathsf{GenerateCircuit}_{h,z}(\{(sk_i, r_{iO,i})\})$ defined below, so that P_1 obtains an obfuscation C' of the circuit $C_{z,sk}$, also defined below.
6. The parties run $\mathsf{Distribute}(C')$.

Offline Output Decryption Phase: Once every party knows C', each party P_i can run $C'(i, ct_{o,i}, \pi_i)$ to obtain y'_i, P_i's padded output under the original MPC protocol. P_i can then compute $y_i \leftarrow y_i \oplus pad_i$.

$\mathsf{CalcSSBHash}(h, \{x_i\}_{i\in[m]})$:

Input: Each party P_i has a key h and x_i. In this protocol we will number the parties starting at 0 (so the first party will be P_0).

Parameters: Let $\lambda \leq s$. Assume h is an SSB hash which has been initialized with fan-in $f = s^{1/2}\lambda/|x|^{1/2}$ and $t = \lceil \log_f m \rceil$.

Before starting: Each party P_i first computes $\leftarrow \mathsf{SSB.Start}(h, x_i)$ to obtain a string $y_{i,0}$ of size λ.

When carrying out the protocol, we will divide the parties into subsets. Let $S_r = \{P_i \mid i \equiv 0 \pmod{f^r}\}$ (and let $S_0 = \{P_i\}_{i\in[m]}$), let the set of children for i in S_r be $D_{i,r} = \{P_j \mid j \equiv 0 \pmod{f^{r-1}}$ and $i \leq j \leq i + f^r\}$, and let the parent of i in S_r be $q_{i,r} = f^r \lfloor i/f^r \rfloor$.

For $k = 1, \ldots, \lceil \log_f m \rceil + 1$, do the following:

Round k: In this round, the parties in the sets S_{k-t}, $t = 1, 3, \ldots, 2\lceil k/2 \rceil - 1$ will participate.

- Each party P_i in S_{k-1} does the following:
  1. If $k - 1 > 0$, receive $y_{j,k-2}$ from each $P_j \in D_{i,k-1}$
  2. If $k - 1 > 0$, calculate $y_{i,k-1} \leftarrow \mathsf{SSB.Combine}(h, \{y_{j,k-2}\}_{P_j \in D_{i,k-1}})$, and send $(y_{i,k-1}, \{y_{j,k-2}\}_{j \in D_{i,k-1}}$ to all parties P_j, $j \in D_{i,k-1}$.
  3. Send $y_{i,k-1}$ to P_{q_i}
- Each party P_j in S_r for $r = 0, \ldots, k-2$ does the following:
  1. Check if received $\pi_{j,r'} = (y_{i,r'}, \{y_{j',r'-1}\}_{j' \in D_{i,r'}})$ from P_{q_j}
  2. If so, append $\pi_{j,r'}$ to π_j.
  3. If $r > 0$, send $\pi_{j,r'}$ to all $P_{j''} \in D_{j,r}$.

For $k' = \lceil \log_f m \rceil + 2, \ldots, 2\lceil \log_f m \rceil + 1$:

Round k': Each party P_j in S_r for $r = 0, \ldots, \lceil \log_f m \rceil$ does the following:

1. Check if received $\pi_{j,r'} = (y_{i,r'}, \{y_{j',r'-1}\}_{j' \in D_{i,r'}})$ from P_{q_j}
2. If so, append $\pi_{j,r'}$ to π_j.
3. If $r > 0$, send $\pi_{j,r'}$ to all $P_{j''} \in D_{j,r}$.

Output: The protocol stops after $2\lceil \log_f m \rceil$ rounds, and every party P_i knows the SSB tree root y and the opening π_i of x_i.

$\mathsf{SSBDistSetup}(l, i^*, \{r_{h,i}\})$:

Parameters: Let $\lambda \leq s$. Let the fan-in f be $s^{1/2}\lambda/l^{1/2}$ and $d = \lceil \log_f m \rceil$.

1. Parties run $\mathsf{Combine}(+, \{r_{h,i}\})$ so that P_1 gets $r_h = \sum r_{h,i}$.
2. P_1 generates an SSB hash key $h \leftarrow \mathsf{SSB.Setup}(1^\lambda, l, d, f, i^*; r_h)$ with l as the block size and i^* as the statistically binding index.

Output: At the end of the protocol, P_1's output is defined as h. All other parties have blank output.

$\mathsf{GenerateCircuit}_{h,z}(\{(sk_i, r_{iO,i})\}_{i \in [m]})$:

Input: P_1 the SSB commitment z; each party P_i has sk_i.

1. Parties run $\mathsf{Combine}(+, \{sk_i\})$ so that P_1 has the master secret key sk.
2. Parties run $\mathsf{Combine}(+, \{r_{iO,i}\})$ so that P_1 has $r_{iO} = \sum r_{iO,i}$.
3. P_1 calculates the obfuscation $C' \leftarrow iO(C_{sk,z}; r_{iO})$.

Output: At the end of the protocol, P_1's output is defined as C'. All other parties have blank output.

Circuit $C_{h,sk,z}(i, ct_{o,i}, r_{o,i}, \pi_i)$:

```
1. If $\mathsf{SSB.Verify}(h, z, i, (ct_{o,i}, r_{o,i}), \pi_i) = 1$:
   (a) Output $\mathsf{TFHE.Dec}_{sk}(ct_{o,i})$.
2. Otherwise, output $\perp$.
```

6.3 Correctness

Correctness of the long-output protocol follows from correctness of the TFHE scheme, the iO scheme, the short-output protocol, the SSB hash, and the CalcSSBHash subprotocol. To see why correctness holds for the CalcSSBHash subprotocol, recall that the end of the protocol each party P_i should know the root label z of the tree along with an opening π_i corresponding to P_i's input x_i. Note that each P_i receives messages for each ancestor of the leaf node $y_{i,0}$ corresponding to P_i's input, and each message is of the form $(y_{i',r}, \{y_{j,r-1}\}_{j \in D_{i',r}})$, containing the ancestor's label and the ancestor's direct children's labels. This is sufficient to learn the root label z of the tree as well as all node labels along the path to $y_{i,0}$ along with the labels of the siblings of all nodes along this path. From this, P_i can onstruct π_i.

6.4 Efficiency

The encrypted MPC and output padding phases of the long-output protocol takes exactly R rounds. The output circuit generation phase consists of two executioons of Distribute, a short-output protocol for SSBDistSetup and GenerateCircuit, and the CalcSSBHash protocol. Distribute takes $\frac{\log m}{\log(s\lambda/l_{out})}$ and the short-output protocol takes $O\left(\frac{\log m}{\log(s\lambda/l_{out})}\right)$.

CalcSSBHash takes $2\lceil \log_f m \rceil$ rounds; it takes $\lceil \log_f m \rceil$ rounds until P_1 knows h, and $\lceil \log_f m \rceil$ additional rounds until all parties P_i know z and π_i. Since $f = s^{1/2}\lambda/l_{out}^{1/2}$, this is $2\lceil \frac{\log m}{\log(s^{1/2}\lambda/l_{out}^{1/2})} \rceil$.

It follows that the total number of rounds used by the long output protocol is $R + O\left(\frac{\log m}{\log(s\lambda/l_{out})}\right)$. Since m is at most a fixed polynomial in λ, this is $O(R)$.

The maximum space used by each party (including messages received and sent) during CalcSSBHash is

$$\begin{aligned} f^2 \cdot l\lambda + c \cdot f \cdot l_{out}\lambda &\leq s\lambda^2/l_{out} \cdot l_{out}\lambda + c \cdot s^{1/2}\lambda/l_{out} \cdot l_{out}\lambda \\ &\leq s \cdot \mathsf{poly}(\lambda). \end{aligned}$$

By the space bounds on the short output protocol and the Distribute protocol, the total space used in the long-output protocol is $s \cdot \mathsf{poly}(\lambda)$.

6.5 Security

Let $I \subset [m]$ be the set of corrupted parties. We describe the behavior of the simulator, which takes as input 1^λ, I, the public key, the parties' outputs $\{y_i\}_{i \in [m]}$, and the corrupted parties' inputs $\{x_i\}_{i \in I}$, and outputs the secret keys and the view of the corrupted parties. Note that as in the short output construction the construction of this simulator allows the corrupted set I to be chosen based on the public key.

Long output protocol simulator:

Input: The simulator receives the corrupted set I, the public key pk, the corrupted parties' inputs and randomness $\{(x_i, r_i)\}_{i\in I}$, and the corrupted parties' outputs $\{y_i\}_{i\in I}$.

Simulated Setup: To generate the corrupted parties' secret keys, the simulator uses the TFHE simulated setup: $(\{sk_{c,i}\}_{i\in I}, \sigma_{sim}) \leftarrow \mathsf{TFHE.Sim.Setup}(pk, I)$.

After initializing the PKI, the simulator carries out a virtual execution of the protocol to generate the corrupted parties' views.

Simulated Encrypted MPC Phase: The simulator performs this phase in exactly the same way as in the short output simulator.

Output Padding Phase: After the R rounds of the encrypted MPC protocol are done, the simulator does the following on behalf of each P_i:

1. Compute a random string $pad_i \in \{0,1\}^{l_{out}}$.
2. If $i \in I$, calculate $ct_{pad_i} \leftarrow \mathsf{TFHE.Enc}_{pk}(pad_i)$; otherwise calculate $ct_{pad_i} \leftarrow \mathsf{TFHE.Enc}_{pk}(0^{l_{out}})$.
3. Calculate $ct_{o,i} \leftarrow \mathsf{TFHE.Eval}(\oplus, ct_{pad_i}, ct_{\sigma_{i,R}})$, the TFHE evaluation of the circuit which pads $\sigma_{i,R}$ with pad_i.

Simulated Output Circuit Generation Phase: At the end of the encrypted execution of the MPC protocol, each party P_i has an encryption $ct_{o,i}$ of their output. The simulator then simulates the output circuit generation phase in the following manner, so that at the end P_1 has an obfuscation of the circuit $\tilde{C}_{h,k,z}$, defined below.

1. The simulator uses the short-output simulator from the previous section for the compiled $\mathsf{SSBDistSetup}$ protocol, where the protocol output is set to be $h \leftarrow \mathsf{SSB.Setup}(1^\lambda, 2l_{out}, f, d, m, r)$ for uniform random r.
2. The simulator runs the protocol $\mathsf{Distribute}(h)$ on behalf of all parties.
3. The simulator chooses a PRF key k.
4. The simulator sets $r_{o,i} = PRF_k(i) \oplus y_i \oplus pad_i$ for all $i \in I$, and $r_{o,i}$ uniformly random for $i \notin I$.
5. The simulator runs the protocol $\mathsf{CalcSSBHash}(h, \{(ct_{o,i}, r_{o,i})\})$ on behalf of all parties, so that each party P_i obtains an SSB commitment z and an opening π_i to $(ct_{o,i}, r_{o,i})$.
6. The simulator uses the short-output simulator from the previous section for the compiled $\mathsf{GenerateCircuit}$ protocol, where the protocol output is set to be the obfuscation $\tilde{C}' = iO(\tilde{C}_{h,k,z})$.
7. The simulator runs $\mathsf{Distribute}(C')$ on behalf of all parties.

Circuit $\tilde{C}_{h,k,z}(i, ct_{o,i}, r_{i,c}, \pi_i)$:

---

1. If $\mathsf{SSB.Verify}(z, i, (ct_{o,i}, r_{i,c}), \pi_i) = 1$:
   (a) Output $r_{i,c} \oplus PRF_k(i)$.
2. Otherwise, output $\perp$.

Claim 3. *Assuming circular simulation security and circular semantic security of the TFHE scheme, security of the SSB hash function, security of the puncturable PRF, and security of the iO scheme, the output of the simulator is indistinguishable from the view of a semi-honest adversary which corrupts all parties in I.*

We prove the claim via a sequence of hybrids, described below.

- **Hybrid H_0:** In this hybrid, the simulator behaves identically to the real world, setting the corrupted parties' secret keys to be the ones generated by the TFHE setup, and running the real-world protocol.
- **Hybrid H_1:** In this hybrid, the simulator behaves in the same way as H_0, except that it uses the TFHE simulator to choose the corrupted parties' secret keys, and it uses the short-output protocol simulator to simulate the executions of $\mathsf{SSBDistSetup}$ and $\mathsf{GenerateCircuit}$.
- **Hybrid H_2:** In this hybrid, the simulator acts in the same way as in H_1, except that samples a PRF key k, sets each party's randomness $r_{o,i} = y_i \oplus PRF_k(i) \oplus pad_i$, and sets the output of the simulated $\mathsf{GenerateCircuit}$ to be $iO(\tilde{C}_{h,z,k})$.
- **Hybrid H_3:** In this hybrid, the simulator acts in the same way as in H_2, except that it sets $ct_{\sigma_{i,0}} \leftarrow \mathsf{TFHE.Enc}_{pk}((0^{|x_i|}, 0^{|r_i|}))$ and $ct_{pad_i} \leftarrow \mathsf{TFHE.Enc}_{pk}(0^{l_{out}})$ for all $i \notin I$.
- **Hybrid H_4:** In this hybrid, the simulator acts in the same way as in H_3, except that it sets $r_{i,o}$ to be uniform random for all $i \notin I$. This hybrid is identical to the ideal world.

We now prove indistinguishability between each successive pair of hybrids.

Claim 4. *Assuming circular simulation security and circular semantic security of the TFHE scheme, the output of the simulator in H_0 is computationally indistinguishable from the output of the simulator in H_1.*

Proof. We prove indistinguishability between H_0 and H_1 via a sequence of two subhybrids:

- **Hybrid $H_{0,1}$:** In this hybrid, the simulator sets the the adversaries' secret keys via the TFHE simulator, and uses the TFHE simulator to generate the honest parties' partial decryptions for the output of $\mathsf{SSBDistSetup}$ and $\mathsf{GenerateCircuit}$.
- **Hybrid $H_{0,2}$:** In this hybrid, the simulator sets the encryptions of each honest party's inputs in $\mathsf{SSBDistSetup}$ and $\mathsf{GenerateCircuit}$ to be encryptions of 0 instead of $r_{h,i}$ and $(sk_i, r_{iO,i})$.

Note that $H_{0,2}$ is identical to H_1. The proofs of indistinguishability between these hybrids is almost identical to the proofs of Claims 1 and 2. For completeness we include them below. □

Claim 5. *Assuming circular simulation security of the TFHE scheme, the output of the simulator in H_0 is indistinguishable from the output of the simulator in $H_{0,1}$.*

Proof. Assume that there is an efficient adversary $\mathcal{A}$ which distinguishes between H_0 and $H_{0,1}$. We use $\mathcal{A}$ to build a reduction $\mathcal{A}'$ against simulation security of the TFHE scheme.

$\mathcal{A}'$ first receives pk from the TFHE challenger. It then sends this to $\mathcal{A}$ and receives back I. $\mathcal{A}'$ then generates random $r_{h,i}$ and $r_{iO,i}$ for each $i \in [m]$, and then sends the plaintexts $\{r_{h,i}\}$ and $\{(\llcorner sk_i \lrcorner, r_{iO,i})\}$ to the challenger along with I. The challenger responds with $\{sk_i\}_{i \in I}$ and ciphertexts corresponding to those plaintexts. $\mathcal{A}'$ then sends the circuit $\mathsf{SSBDistSetup}$ as a query to the challenger, receives the partial decryptions of the honest parties, and runs $\mathsf{TFHE.CombineParts}$ and $\mathsf{TFHE.Round}$ to obtain h. It then runs the $\mathsf{CalcSSBHash}$ protocol on behalf of all parties to obtain z, and then submits the circuit $\mathsf{GenerateCircuit}_{h,z}$ as its second query to the challenger. Once it receives the honest parties' partial decryptions from the challenger, $\mathcal{A}'$ generates the views of the corrupted parties, and sends this view to $\mathcal{A}$. It outputs the output of $\mathcal{A}$.

If the TFHE simulation security challenger enacts the real-world experiment, then the view of $\mathcal{A}$ is identical to H_0. If the TFHE challenger enacts the ideal-world experiment then the view of $\mathcal{A}$ is identical to $H_{0,1}$. This means that $\mathcal{A}'$ is an efficient distinguisher between the real and ideal experiments of the TFHE simulation game. □

Claim 6. *Assuming circular semantic security of the TFHE scheme, the output of the simulator in $H_{0,1}$ is indistinguishable from the output of the simulator in $H_{0,2}$.*

Proof. Assume that there is an efficient adversary $\mathcal{A}$ which distinguishes between $H_{0,1}$ and $H_{0,2}$ with non-negligible probability. We use $\mathcal{A}$ to build a reduction $\mathcal{A}'$ against circular semantic security of the TFHE scheme.

$\mathcal{A}'$ performs the same steps as the simulator in $H_{0,1}$, except that instead of computing the honest parties' ciphertext inputs for the $\mathsf{SSBDistSetup}$ and $\mathsf{GenerateCircuit}$ protocols it sends the plaintexts $\{r_{h,i}\}$ and $\{(\llcorner sk_i \lrcorner, r_{iO,i})\}$ to the challenger, and uses the ciphertexts received from the challenger in place of the honest parties' ciphertexts. (Recall that the challenger replaces the symbol $\llcorner sk_i \lrcorner$ with the honest party's secret key sk_i. $\mathcal{A}'$ sends the view of the corrupted parties to $\mathcal{A}$, and outputs the output of $\mathcal{A}$.

If the TFHE circular semantic security challenger sends ciphertexts with the true values, then the view of $\mathcal{A}$ is identical to $H_{0,1}$. If the challenger sends encryptions of 0 then the view of $\mathcal{A}$ is identical to $H_{0,2}$. This means that $\mathcal{A}'$ is a successful efficient adversary against the TFHE circular semantic security. □

Claim 7. *Assuming security of the SSB hash function, the puncturable PRF, and the iO scheme, the output of the simulator in H_1 is computationally indistinguishable from the output of the simulator in H_2.*

Proof. We prove indistinguishability between H_1 and H_2 via a sequence of two subhybrids $H_{1,j}$ for $j \in [m]$, where $H_{1,0} = H_1$ and $H_{1,m} = H_2$. In $H_{1,j}$, the simulator sets the output of the simulated $\mathsf{GenerateCircuit}$ protocol to be the circuit $C' = iO(\tilde{C}^{j}_{sk,k,z})$, where $\tilde{C}^{j}_{h,sk,k,z})$ is defined below. We show indistinguishability between each successive pair $H_{1,j-1}$ and $H_{1,j}$ by the following sequence of hybrids:

- **Hybrid** $H_{1,j-1,1}$**:** In this hybrid, the simulator behaves the same as in $H_{1,j-1}$ except that it sets the output of the simulated $\mathsf{SSBDistSetup}$ protocol to be $h \leftarrow \mathsf{SSB.Setup}(1^\lambda, l, j)$, where the binding index $i^* = j$.
- **Hybrid** $H_{1,j-1,2}$**:** In this hybrid, the simulator behaves the same as in $H_{1,j-1,1}$, except that it uses an obfuscation of the circuit $\tilde{C}^{j,2}_{h,sk,k',z,y_j}$ in the output of $\mathsf{GenerateCircuit}$, with a punctured PRF key k' punctured at j.
- **Hybrid** $H_{1,j-1,3}$**:** In this hybrid, the simulator behaves the same as in $H_{1,j-1,2}$, except that it sets the randomness $r_{o,j}$ of party P_j to $y_j \oplus PRF_k(j) \oplus pad_i$.
- **Hybrid** $H_{1,j-1,4}$**:** In this hybrid, the simulator behaves the same as in $H_{1,j-1,2}$, except that it uses an obfuscation of the circuit $\tilde{C}^{j}_{h,sk,k,z}$ when computing the output of the simulated $\mathsf{GenerateCircuit}$ protocol. This hybrid is identical to $H_{1,j}$.

Circuit $\tilde{C}^{j}_{sk,k,z}(i, ct_i, r_{i,c}, \pi_i)$:

```
1. If $\mathsf{SSB.Verify}(h, z, i, (ct_i, r_{i,c}), \pi_i) = 1$:
   (a) If $i \leq j$, output $r_{i,c} \oplus PRF_k(i)$.
   (b) Otherwise, output $\mathsf{TFHE.Dec}_{sk}(ct_i)$.
2. Otherwise, output $\perp$.
```

Circuit $\tilde{C}^{j,2}_{h,sk,k',z,out_i}(i, ct_i, r_{i,c}, \pi_i)$:

```
1. If $\mathsf{SSB.Verify}(h, z, i, (ct_i, r_{i,c}), \pi_i) = 1$:
```

   (a) If $i < j$, output $r_{i,c} \oplus PRF_k(i)$.
   (b) If $i = j$, output out_i.
   (c) Otherwise, output $\mathsf{TFHE.Dec}_{sk}(ct_i)$.
2. Otherwise, output $\perp$.

We prove indistinguishability of each successive pair of these hybrids below. □

Claim 8. *Assuming the index hiding property of the SSB hash function, the output of the simulator in $H_{1,j-1}$ is indistinguishable from the output of the simulator in $H_{1,j-1,1}$.*

Proof. Assume that there is an efficient adversary $\mathcal{A}$ which distinguishes between $H_{1,j-1}$ and $H_{1,j-1,1}$ with non-negligible probability. We use $\mathcal{A}$ to build a reduction $\mathcal{A}'$ against the index hiding property of the SSB hash function.

$\mathcal{A}'$ interacts with a challenger which supplies $\mathcal{A}'$ with an SSB hash key with the statistical binding bit i^* set to either $j-1$ or j. $\mathcal{A}'$ performs the same steps as the simulator in $H_{1,j-1}$, except that it queries the challenger for the SSB hash key h, which it uses as the simulated output of the $\mathsf{SSBDistSetup}$ protocol. It sends the view of the corrupted parties to $\mathcal{A}$, and outputs the output of $\mathcal{A}$.

If the SSB index hiding challenger sends an SSB hash key with the statistical binding index i^* set to $j-1$, then the view of $\mathcal{A}$ is identical to $H_{1,j-1}$. If the challenger sends an SSB hash key with i^* set to j, the view of $\mathcal{A}$ is identical to $H_{1,j-1,1}$. This means that $\mathcal{A}'$ is a successful efficient adversary against the index hiding property of the SSB hash function. □

Claim 9. *Assuming security of the iO scheme and statistical binding of the SSB hash function, the output of the simulator in $H_{1,j-1,1}$ is indistinguishable from the output of the simulator in $H_{1,j-1,2}$.*

Proof. Assume that there is an efficient adversary $\mathcal{A}$ which distinguishes between $H_{1,j-1,1}$ and $H_{1,j-1,2}$ with non-negligible probability. We use $\mathcal{A}$ to build a reduction $\mathcal{A}'$ against the security of the iO scheme.

$\mathcal{A}'$ interacts with a challenger, which receives two circuits from $\mathcal{A}'$ and sends the obfuscation of one of the two circuits. $\mathcal{A}'$ performs the same steps as the simulator in $H_{1,j-1,1}$, except that instead of computing the simulated output for $\mathsf{GenerateCircuit}$ directly, it sends the two circuits $C_0 = \tilde{C}^{j-1}_{h,sk,k,z}$ and $C_1 = \tilde{C}^{j,2}_{h,sk,k',z}$ to the challenger and receives back an obfuscation of one of them, which it uses as the simulated output of the $\mathsf{SSBDistSetup}$ protocol. It sends the view of the corrupted parties to $\mathcal{A}$, and outputs the output of $\mathcal{A}$.

If the iO challenger sends an obfuscation of C_0, then the view of $\mathcal{A}$ is identical to $H_{1,j-1,1}$. If the challenger sends obfuscation of C_1, the view of $\mathcal{A}$ is identical to $H_{1,j-1,2}$. This means that $\mathcal{A}'$ is a successful efficient adversary against the iO challenger, provided that C_0 and C_1 are functionally equivalent. Note that the only input where the behavior of C_1 could differ from the behavior of C_0 is on inputs (j, ct, r, π) , where $ct \neq ct_{o,j}$, the committed output ciphertext of party j, and $\mathsf{SSB.Verify}(h, z, j, (ct, r), \pi) = 1$. But by the statistical binding property, this is impossible. □

Claim 10. *Assuming security of the puncturable PRF, the output of the simulator in $H_{1,j-1,2}$ is indistinguishable from the output of the simulator in $H_{1,j-1,3}$.*

Proof. Assume that there is an efficient adversary $\mathcal{A}$ which distinguishes between $H_{1,j-1,2}$ and $H_{1,j-1,3}$ with non-negligible probability. We use $\mathcal{A}$ to build a reduction $\mathcal{A}'$ against security of the puncturable PRF.

$\mathcal{A}'$ interacts with a challenger which supplies $\mathcal{A}'$ with a PRF key k' punctured at position j, and either the evaluation $PRF_k(j)$ with the unpunctured key k or a uniform random string of the same length. $\mathcal{A}'$ performs the same steps as the simulator in $H_{1,j-1}$, except that it uses the challenger's key k' when computing the circuit $\tilde{C}^{j,2}_{h,sk,k',y,z_j}$, and sets $r_{o,j} = y_j + a$, where a is the response from the challenger. It sends the view of the corrupted parties to $\mathcal{A}$, and outputs the output of $\mathcal{A}$.

If the challenger sends $a = PRF_k(j)$, then the view of $\mathcal{A}$ is identical to $H_{1,j-1,3}$. If the challenger sends a uniform random a, the view of $\mathcal{A}$ is identical to $H_{1,j-1,1,2}$. This means that $\mathcal{A}'$ is a successful efficient adversary against security of the puncturable PRF. □

Claim 11. *Assuming security of the iO scheme and the statistical binding of the SSB hash function, the output of the simulator in $H_{1,j-1,3}$ is indistinguishable from the output of the simulator in $H_{1,j-1,4}$.*

Proof. Assume that there is an efficient adversary $\mathcal{A}$ which distinguishes between $H_{1,j-1,3}$ and $H_{1,j-1,4}$ with non-negligible probability. We use $\mathcal{A}$ to build a reduction $\mathcal{A}'$ against the security of the iO scheme.

$\mathcal{A}'$ interacts with a challenger, which receives two circuits from $\mathcal{A}'$ and sends the obfuscation of one of the two circuits. $\mathcal{A}'$ performs the same steps as the simulator in $H_{1,j-1,3}$, except that instead of computing the simulated output for $\mathsf{GenerateCircuit}$ directly, it sends the two circuits $C_0 = \tilde{C}^{j,2}_{h,sk,k',z}$ and $C_1 = \tilde{C}^{j}_{h,sk,k,z}$ to the challenger and receives back an obfuscation of one of them, which it uses as the simulated output of the $\mathsf{SSBDistSetup}$ protocol. It sends the view of the corrupted parties to $\mathcal{A}$, and outputs the output of $\mathcal{A}$.

If the iO challenger sends an obfuscation of C_0, then the view of $\mathcal{A}$ is identical to $H_{1,j-1,3}$. If the challenger sends obfuscation of C_1, the view of $\mathcal{A}$ is identical to $H_{1,j-1,4}$. This means that $\mathcal{A}'$ is a successful efficient adversary against the iO challenger, provided that C_0 and C_1 are functionally equivalent. Note that the only input where the behavior of C_1 could differ from the behavior of C_0 is on inputs (j, ct, r, π) , where $r \neq r_{o,j}$, the committed randomness of party j, and $\mathsf{SSB.Verify}(h, z, j, (ct, r), \pi) = 1$. But by the statistical binding property, this is impossible. □

Claim 12. *Assuming semantic security of the TFHE scheme, the output of the simulator in H_2 is computationally indistinguishable from the output of the simulator in H_3.*

Proof. Assume that there is an efficient adversary $\mathcal{A}$ which distinguishes between H_2 and H_3 with non-negligible probability. We use $\mathcal{A}$ to build a reduction $\mathcal{A}'$ against semantic security of the TFHE scheme.

$\mathcal{A}'$ performs the same steps as the simulator in H_2, except that instead of encrypting the honest parties' MPC inputs and randomness directly it sends the plaintexts $\{(x_i, r_i)\}_{i \notin I}$ and $\{pad_i\}_{i \notin I}$ to the challenger, and uses the ciphertexts received from the challenger as $\{ct_{\sigma_{i,0}}\}_{i \notin I}$ and $\{ct_{pad_i}\}_{i \notin I}$. It sends the view of the corrupted parties to $\mathcal{A}$, and outputs the output of $\mathcal{A}$.

If the TFHE circular semantic security challenger sends ciphertexts with the true values, then the view of $\mathcal{A}$ is identical to H_2. If the challenger sends encryptions of 0 then the view of $\mathcal{A}$ is identical to H_3. This means that $\mathcal{A}'$ is a successful efficient adversary against the TFHE semantic security. □

Claim 13. *The output of the simulator in H_3 is identical to the output of the simulator in H_4.*

Proof. This follows from the fact that the distribution of a one-time pad is identical to uniform. □

Acknowledgments

We would like to thank Shir Maimon and Wei-Kai Lin for helpful discussions. We gratefully acknowledge the TCC '20 reviewers for their thoughtful comments. We would like to thank Tatsuaki Okamoto for being supportive of this work. We thank Manoj Prabhakaran for pointing out the related work by Boyle, Jain, Prabhakaran, and Yu [25].

References

[1] Ahn, K.J., Guha, S.: Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. ACM Transactions on Parallel Computing (TOPC) **4**(4), 17 (2018)

[2] Ananth, P., Chen, Y., Chung, K., Lin, H., Lin, W.: Delegating RAM computations with adaptive soundness and privacy. In: Theory of Cryptography - 14th International Conference, TCC. pp. 3–30 (2016)

[3] Andoni, A., Nikolov, A., Onak, K., Yaroslavtsev, G.: Parallel algorithms for geometric graph problems. In: Symposium on Theory of Computing, STOC. pp. 574–583 (2014)

[4] Andoni, A., Song, Z., Stein, C., Wang, Z., Zhong, P.: Parallel graph connectivity in log diameter rounds. In: 59th IEEE Annual Symposium on Foundations of Computer Science, FOCS. pp. 674–685 (2018)

[5] Andoni, A., Stein, C., Zhong, P.: Log diameter rounds algorithms for 2-vertex and 2-edge connectivity. In: 46th International Colloquium on Automata, Languages, and Programming, ICALP. pp. 14:1–14:16 (2019)

[6] Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: EUROCRYPT. pp. 483–501 (2012)

[7] Assadi, S.: Simple round compression for parallel vertex cover. CoRR **abs/1709.04599** (2017)

[8] Assadi, S., Bateni, M., Bernstein, A., Mirrokni, V.S., Stein, C.: Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA. pp. 1616–1635 (2019)

[9] Assadi, S., Khanna, S.: Randomized composable coresets for matching and vertex cover. In: Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA. pp. 3–12 (2017)

[10] Assadi, S., Sun, X., Weinstein, O.: Massively parallel algorithms for finding well-connected components in sparse graphs. In: ACM Symposium on Principles of Distributed Computing, PODC. pp. 461–470 (2019)

[11] Badrinarayanan, S., Jain, A., Manohar, N., Sahai, A.: Threshold multi-key FHE and applications to round-optimal MPC. IACR Cryptology ePrint Archive **2018**, 580 (2018)

[12] Bahmani, B., Kumar, R., Vassilvitskii, S.: Densest subgraph in streaming and mapreduce. Proceedings of the VLDB Endowment **5**(5), 454–465 (2012)

[13] Bahmani, B., Moseley, B., Vattani, A., Kumar, R., Vassilvitskii, S.: Scalable k-means++. Proceedings of the VLDB Endowment **5**(7), 622–633 (2012)

[14] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. J. ACM **59**(2), 6:1–6:48 (2012)

[15] Bateni, M., Bhaskara, A., Lattanzi, S., Mirrokni, V.: Distributed balanced clustering via mapping coresets. In: Advances in Neural Information Processing Systems. pp. 2591–2599 (2014)

[16] Behnezhad, S., Brandt, S., Derakhshan, M., Fischer, M., Hajiaghayi, M., Karp, R.M., Uitto, J.: Massively parallel computation of matching and MIS in sparse graphs. In: ACM Symposium on Principles of Distributed Computing, PODC. pp. 481–490 (2019)

[17] Behnezhad, S., Derakhshan, M., Hajiaghayi, M., Karp, R.M.: Massively parallel symmetry breaking on sparse graphs: MIS and maximal matching. CoRR **abs/1807.06701** (2018)

[18] Behnezhad, S., Hajiaghayi, M., Harris, D.G.: Exponentially faster massively parallel maximal matching. In: 60th IEEE Annual Symposium on Foundations of Computer Science, FOCS. pp. 1637–1649 (2019)

[19] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Proceedings of the 20th Annual ACM Symposium on Theory of Computing, STOC. pp. 1–10 (1988)

[20] Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M.R., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. In: Advances in Cryptology - CRYPTO. pp. 565–596 (2018)

[21] Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Advances in Cryptology - ASIACRYPT. pp. 280–300 (2013)

[22] Boyle, E., Chung, K., Pass, R.: Large-scale secure computation: Multi-party computation for (parallel) RAM programs. In: Advances in Cryptology - CRYPTO. pp. 742–762 (2015)

[23] Boyle, E., Chung, K., Pass, R.: Oblivious parallel RAM and applications. In: Theory of Cryptography - 13th International Conference, TCC. pp. 175–204 (2016)

[24] Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Public-Key Cryptography - PKC. pp. 501–519 (2014)

[25] Boyle, E., Jain, A., Prabhakaran, M., Yu, C.H.: The Bottleneck Complexity of Secure Multiparty Computation. In: 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018). pp. 24:1–24:16 (2018)

[26] Brakerski, Z., Perlman, R.: Lattice-based fully dynamic multi-key FHE with short ciphertexts. In: Advances in Cryptology - CRYPTO. pp. 190–213 (2016)

[27] Chan, T.H., Chung, K., Lin, W., Shi, E.: MPC for MPC: secure computation on a massively parallel computing architecture. In: 11th Innovations in Theoretical Computer Science Conference, ITCS. pp. 75:1–75:52 (2020)

[28] Chan, T.H., Chung, K., Shi, E.: On the depth of oblivious parallel RAM. In: Advances in Cryptology - ASIACRYPT. pp. 567–597 (2017)

[29] Chan, T.H., Nayak, K., Shi, E.: Perfectly secure oblivious parallel RAM. In: Theory of Cryptography - 16th International Conference, TCC 2018. pp. 636–668 (2018)

[30] Chan, T.H., Shi, E.: Circuit OPRAM: unifying statistically and computationally secure orams and oprams. In: Theory of Cryptography - 15th International Conference, TCC. pp. 72–107 (2017)

[31] Chang, Y., Fischer, M., Ghaffari, M., Uitto, J., Zheng, Y.: The complexity of $(\Delta{+}1)$ coloring in congested clique, massively parallel computation, and centralized local computation. In: ACM Symposium on Principles of Distributed Computing, PODC. pp. 471–480 (2019)

[32] Chen, Y., Chow, S.S.M., Chung, K., Lai, R.W.F., Lin, W., Zhou, H.: Cryptography for parallel RAM from indistinguishability obfuscation. In: ACM Conference on Innovations in Theoretical Computer Science, ITCS. pp. 179–190 (2016)

[33] Chung, K.M., Qian, L.: Adaptively secure garbling schemes for parallel computations. In: TCC (2019)

[34] Czumaj, A., Łącki, J., Mądry, A., Mitrović, S., Onak, K., Sankowski, P.: Round compression for parallel matching algorithms. In: Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC. pp. 471–484 (2018)

[35] Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Advances in Cryptology - CRYPTO. pp. 93–122 (2016)

[36] Ene, A., Im, S., Moseley, B.: Fast clustering using mapreduce. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 681–689. ACM (2011)

[37] Ene, A., Nguyen, H.: Random coordinate descent methods for minimizing decomposable submodular functions. In: International Conference on Machine Learning. pp. 787–795 (2015)

[38] Gamlath, B., Kale, S., Mitrovic, S., Svensson, O.: Weighted matchings via unweighted augmentations. In: ACM Symposium on Principles of Distributed Computing, PODC. pp. 491–500 (2019)

[39] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS. pp. 40–49 (2013)

[40] Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: CRYPTO 2013. pp. 75–92 (2013). https://doi.org/10.1007/978-3-642-40041-4_5, https://doi.org/10.1007/978-3-642-40041-4_5

[41] Ghaffari, M., Gouleakis, T., Konrad, C., Mitrovic, S., Rubinfeld, R.: Improved massively parallel computation algorithms for mis, matching, and vertex cover. In: ACM Symposium on Principles of Distributed Computing, PODC. pp. 129–138 (2018)

[42] Ghaffari, M., Lattanzi, S., Mitrović, S.: Improved parallel algorithms for density-based network clustering. In: International Conference on Machine Learning. pp. 2201–2210 (2019)

[43] Ghaffari, M., Uitto, J.: Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA. pp. 1636–1653 (2019)

[44] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: 25th Annual Symposium on Foundations of Computer Science, FOCS. pp. 464–479 (1984)

[45] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing, STOC. pp. 218–229 (1987)

[46] Hubáček, P., Wichs, D.: On the communication complexity of secure function evaluation with long output. In: ITCS. pp. 163–172 (2015)

[47] Im, S., Moseley, B., Sun, X.: Efficient massively parallel methods for dynamic programming. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC. pp. 798–811 (2017)

[48] Jain, A., Rasmussen, P.M.R., Sahai, A.: Threshold fully homomorphic encryption. IACR Cryptol. ePrint Arch. **2017**, 257 (2017)

[49] Karloff, H.J., Suri, S., Vassilvitskii, S.: A model of computation for mapreduce. In: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA. pp. 938–948 (2010)

[50] Katz, J., Ostrovsky, R., Smith, A.D.: Round efficiency of multi-party computation with a dishonest majority. In: EUROCRYPT. pp. 578–595 (2003)

[51] Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS. pp. 669–684 (2013)

[52] Kumar, R., Moseley, B., Vassilvitskii, S., Vattani, A.: Fast greedy algorithms in mapreduce and streaming. TOPC **2**(3), 14:1–14:22 (2015)

[53] Łącki, J., Mirrokni, V.S., Wlodarczyk, M.: Connected components at scale via local contractions. CoRR **abs/1807.10727** (2018)

[54] Lattanzi, S., Moseley, B., Suri, S., Vassilvitskii, S.: Filtering: a method for solving graph problems in mapreduce. In: SPAA. pp. 85–94 (2011)

[55] López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the 44th Symposium on Theory of Computing Conference, STOC. pp. 1219–1234 (2012)

[56] Lu, S., Ostrovsky, R.: Black-box parallel garbled ram. In: CRYPTO 2017. pp. 66–92 (2017)

[57] Mirrokni, V.S., Zadimoghaddam, M.: Randomized composable core-sets for distributed submodular maximization. In: STOC. pp. 153–162 (2015)

[58] Mirzasoleiman, B., Karbasi, A., Sarkar, R., Krause, A.: Distributed submodular maximization: Identifying representative elements in massive data. In: Advances in Neural Information Processing Systems. pp. 2049–2057 (2013)

[59] Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Fischlin, M., Coron, J. (eds.) EUROCRYPT. pp. 735–763 (2016)

[60] Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: EUROCRYPT. pp. 735–763 (2016)

[61] Nayak, K., Wang, X.S., Ioannidis, S., Weinsberg, U., Taft, N., Shi, E.: GraphSC: Parallel Secure Computation Made Easy. In: IEEE S & P (2015)

[62] Onak, K.: Round compression for parallel graph algorithms in strongly sublinear space. CoRR **abs/1807.08745** (2018)

[63] Parter, M., Yogev, E.: Distributed algorithms made secure: A graph theoretic approach. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA. pp. 1693–1710 (2019)

[64] Parter, M., Yogev, E.: Secure distributed computing made (nearly) optimal. pp. 107–116. PODC '19 (2019)

[65] Pass, R.: Bounded-concurrent secure multi-party computation with a dishonest majority. In: Babai, L. (ed.) STOC. pp. 232–241. ACM (2004)

[66] Peikert, C., Shiehian, S.: Multi-key FHE from lwe, revisited. In: TCC 2016. pp. 217–238 (2016)

[67] da Ponte Barbosa, R., Ene, A., Nguyen, H.L., Ward, J.: A new framework for distributed submodular maximization. In: FOCS. pp. 645–654 (2016)

[68] Rastogi, V., Machanavajjhala, A., Chitnis, L., Sarma, A.D.: Finding connected components in map-reduce in logarithmic rounds. In: 29th IEEE International Conference on Data Engineering, ICDE. pp. 50–61 (2013)

[69] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6), 34:1–34:40 (2009)

[70] Roughgarden, T., Vassilvitskii, S., Wang, J.R.: Shuffles and circuits: (on lower bounds for modern parallel computation). In: Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA. pp. 1–12 (2016)

[71] Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: Symposium on Theory of Computing, STOC. pp. 475–484 (2014)

[72] Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: STOC. pp. 475–484. ACM (2014)

[73] Yaroslavtsev, G., Vadapalli, A.: Massively parallel algorithms and hardness for single-linkage clustering under ℓ_p-distances. In: Proceedings of the 35th International Conference on Machine Learning (2018)