MPC for MPC: Secure Computation on a Massively Parallel Computing Architecture

T-H. Hubert Chan
The University of Hong Kong, Hong Kong
hubert@cs.hku.hk

Kai-Min Chung
Academia Sinica, Taipei City, Taiwan
kmchung@iis.sinica.edu.tw

Wei-Kai Lin
Cornell University, Ithaca, NY, USA
wklin@cs.cornell.edu

Elaine Shi
Cornell University, Ithaca, NY, USA
elaine@cs.cornell.edu

Abstract

Massively Parallel Computation (MPC) is a model of computation widely believed to best capture realistic parallel computing architectures such as large-scale MapReduce and Hadoop clusters. Motivated by the fact that many data analytics tasks performed on these platforms involve sensitive user data, we initiate the theoretical exploration of how to leverage MPC architectures to enable efficient, privacy-preserving computation over massive data. Clearly if a computation task does not lend itself to an efficient implementation on MPC even without security, then we cannot hope to compute it efficiently on MPC with security. We show, on the other hand, that *any task that can be efficiently computed on MPC can also be securely computed with comparable efficiency*. Specifically, we show the following results:

- any MPC algorithm can be compiled to a *communication-oblivious* counterpart while asymptotically *preserving its round and space complexity*, where communication-obliviousness ensures that any network intermediary observing the communication patterns learn no information about the secret inputs;
- assuming the existence of Fully Homomorphic Encryption with a suitable notion of compactness and other standard cryptographic assumptions, any MPC algorithm can be compiled to a secure counterpart that defends against an adversary who controls not only intermediate network routers but additionally up to $1/3 - \eta$ fraction of machines (for an arbitrarily small constant η) – moreover, this compilation preserves the round complexity tightly, and preserves the space complexity upto a multiplicative security parameter related blowup.

As an initial exploration of this important direction, our work suggests new definitions and proposes novel protocols that blend algorithmic and cryptographic techniques.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols

Keywords and phrases massively parallel computation, secure multi-party computation

Digital Object Identifier 10.4230/LIPIcs.ITCS.2020.75

Funding *T-H. Hubert Chan*: T-H. Hubert Chan was partially supported by the Hong Kong RGC under the grant 17200418.
Kai-Min Chung: This research is partially supported by the Academia Sinica Career Development Award under Grant no. 23-17 and Ministry of Science and Technology, Taiwan, under Grant no. MOST 106-2628-E-001-002-MY3.
Wei-Kai Lin: This work is supported by the DARPA Brandeis award.
Elaine Shi: This work is supported in part by NSF CNS-1453634, an ONR YIP award, a Packard Fellowship, and an IARPA HECTOR grant under a subcontract from IBM.

11th Innovations in Theoretical Computer Science Conference (ITCS 2020).
Editor: Thomas Vidick; Article No. 75; pp. 75:1–75:52

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We gratefully thank Xiaorui Sun for his patient and detailed explanations about the Massively Parallel Computation (MPC) model, for answering many of our technical questions, and for suggesting an idea to transform an MPC protocol without the s-sender-constraint to one that has.

1 Introduction

In the past decade, parallel computation has been widely adopted to manipulate and analyze large-scale data-sets, and numerous programming paradigms such as MapReduce, Hadoop, and Spark have been popularized to help program large computing clusters. This has partly served as a driving force for the algorithms community to better understand the power and limitations of such parallel computation models. The first theoretic model capturing modern parallel computation frameworks was proposed by Karloff, Suri, and Vassilvitskii [79]. Since then, a flurry of results have appeared proposing refinements to the model as well as novel algorithms with better asymptotical and practical efficiency [4, 40, 76, 79, 80, 83, 89, 99].

With these amazing efforts, the community has converged on a model called *Massively Parallel Computation* (MPC), which is believed to best capture large computing clusters (e.g., those operated by companies like Google and Facebook) consisting of a network of Random-Access Machines (RAMs), each with a somewhat considerable amount of local memory and processing power – and yet each individual machine is not powerful enough to store the massive amount of data available. In the MPC model of computation, we typically assume a total of N data records where N is rather large (in practice, the data size can range from tens of terabytes to a petabyte). Each machine can locally store only $s = N^{\varepsilon}$ amount of data for some constant $\varepsilon \in (0, 1)$; and the number of machines $m \geq N^{1-\varepsilon}$ such that all machines can jointly store the entire dataset. In many MPC algorithms it is also desirable if $m \cdot s = \widetilde{O}(N)$ or $m \cdot s \leq N^{1+\theta}$ for some small constant $\theta \in (0, 1)$, i.e., the total space consumed should not be too much larger than the dataset itself [2, 5, 79, 83].

In the standard algorithms literature on MPC, a key focus has been the design of algorithms that minimize the *round complexity*, partly by harnessing the reasonably large local memory that is available on each processing unit. Using round complexity as a primary metric, a rich set of computational tasks have been investigated in the MPC model, including graph problems [2, 4–6, 10–13, 15, 19, 20, 29, 36, 40, 52, 60, 63, 81, 83, 93, 97], clustering [16, 17, 46, 61, 107] and submodular function optimization [41, 47, 80, 90]. Interestingly, it is also known that a number of tasks (such as sorting, parity, minimum spanning tree) that either suffered from an almost logarithmic depth lower bound on a classical Parallel Random-Access Machine (PRAM) now can be accomplished in $O(1)$ or sublogarithmic number of rounds on an MPC framework [40, 67, 79, 93]. Note that a PRAM assumes that each processing unit has $O(1)$ local storage and thus PRAM is not the best fit for capturing modern parallel computing clusters.

1.1 Privacy-Preserving Data Analytics on MPC Frameworks

In this paper, we are the first to ask the question, how can we leverage an MPC cluster to facilitate *privacy-preserving*, large-scale data analytics? This question is of increasing importance because numerous data analytics tasks we want to perform on these frameworks involve sensitive user data, e.g., users' behavior history on websites and/or social networks, medical records, or genomic data. We consider two primary scenarios:

Scenario 1: MPC with secure end-points

We may consider an MPC framework where the end-points[1] are secured by trusted processors such as Intel SGX. Without loss of generality, we may assume that data is encrypted in memory or in transit such that all the secure processors can decrypt them inside a hardware-enabled sandbox where computation will take place (to achieve this the secure processors may employ standard cryptographic techniques to perform a secure handshake and establish a shared encryption key). Finally, when the computation result is ready, the secure processors can send encrypt results to an authorized analyst who has a corresponding key to decrypt the results.

In this scenario, we consider a network adversary (e.g., compromised operating systems, intermediate network routers, or system administrators) that can observe the communication patterns between the end-points, and we would like to make sure that the MPC algorithm's *communication pattern leak no information* about the secret data.

Note that in Scenario 1, we make a simplifying assumption that the adversary cannot observe the memory access patterns on end-points: since known works on efficient Oblivious RAM (ORAM) [64, 66, 100, 102] have in principle solved this problem; not only so, in recent work the first secure processor with ORAM support has been taped out [50, 51, 98].

Scenario 2: MPC with insecure end-points

In the second scenario, imagine that the adversary controls not only the intermediate network routers but also some of the end-points. For example, the end-points may be legacy machines without trusted hardware support, and they may have a comprised operating system. The adversary may also be a rogue system administrator who has access to a subset of the machines. We assume, however, that the adversary controls only a small subset of the end-points – such an assumption is reasonable if the end-points are hosted by various organizations, or by the same organization but in different data centers, or if they have diverse hardware/software configurations such that they are unlikely to be all hit by the same virus.

In this scenario, we would like to obtain a guarantee similar to that of cryptographic Secure Multi-Party Computation (SMPC)[2], i.e., an adversary controlling a relatively small subset of the machines cannot learn more information beyond what is implied by the union of the corrupt machine's outputs. Note that in this scenario, all machines' outputs can also be in encrypted format such that only an authorized data analyst can decrypt the result; or alternatively, secret shared such that only the authorized data analyst can reconstruct – in this case, the adversary should not be able to learn anything at all from the computation.

With the aforementioned scenarios in mind, we ask, what computation tasks can be securely and efficiently computed on an MPC architecture? Clearly, if a computation task does not lend itself to efficient computation on MPC *even without security*, we cannot hope to attain an efficient and secure solution on the same architecture. Therefore, the best we can hope for is the following: for computational tasks that indeed have efficient MPC algorithms, we now want to compute the same task securely on MPC while preserving the efficiency of the original insecure algorithm. In other words, we ask the following important question:

> Can we *securely* evaluate a function f on an MPC framework, while paying not too much more overhead than evaluating f *insecurely* on MPC?

[1] By end-points, we mean the machines, as opposed to the communication/network.
[2] In this paper, to avoid confusion, we use SMPC to mean cryptographic Secure Multi-Party Computation; and we use MPC to mean Massively Parallel Computation.

1.2 Our Results and Contributions

Conceptual and definitional contributions

We initiate the exploration of how to leverage Massively Parallel Computation (MPC) to secure large-scale computation. The widespread empirical success of MPC frameworks in practice, as well as the typically sensitive nature of the data involved in the analytics provide strong motivations for our exploration. We hope that the formal definitions and theoretical feasibility results in our work will encourage future work along this direction, and hopefully leading to practical solutions for privacy-preserving large-scale data analytics.

In comparison, although earlier works originating from the cryptography community have explored secure computation on parallel architectures, most known results [3, 24, 25, 32, 34, 35, 37, 38, 87, 92] adopt PRAM as the model of computation. As discussed later in Section 2, known results specialized for PRAMs do not directly lead to the type of results in this paper due to the discrepancy both in model and in metrics. As mentioned, the PRAM model is arguably a mismatch for the parallel computing architectures encountered in most practical scenarios. This is exactly why in the past decade, the algorithms community have focused more on the modern MPC model which better captures the massively parallel computing clusters deployed by companies like Google and Facebook. Therefore, we hope that our work will bring the MPC model to the attention of the cryptography community for exploring parallel secure computation.

We proceed to present a summary of our major results.

Communication-oblivious MPC

To securely compute a function in Scenario 1 and as a stepping stone towards enabling Scenario 2, we first define a notion of *communication obliviousness* for MPC algorithms. Informally speaking, we want that the adversary learns no information about the secret inputs after observing an MPC algorithm's communication patterns. In this paper we require a very strong notion of communication obliviousness, where we simply want that the MPC algorithm's communication patterns be deterministic and input independent[3].

We prove that any MPC algorithm Π can be compiled to a communication-oblivious counterpart while asymptotically preserving its round complexity and space consumption.

▶ **Theorem 1** (Communication-oblivious MPC algorithms). *Suppose that $s = N^\varepsilon$ and that m is upper bounded by a fixed polynomial in N. Given any MPC algorithm Π that completes in R rounds where each of the m machines has s local space, there is a communication-oblivious MPC algorithm $\widetilde{\Pi}$ that computes the same function as Π except with $\exp(-\Omega(\sqrt{s}))$ probability, and moreover $\widetilde{\Pi}$ completes in $O(R)$ rounds, and consuming $O(s)$ space on each of the m machines. Furthermore, only $O(m \cdot s)$ amount of data are communicated in each round in an execution of $\widetilde{\Pi}$.*

Note that numerous interesting MPC algorithms known thus far have total communication at least $\Omega(R \cdot m \cdot s)$ where R denotes the protocol's round complexity (ignoring polylogarithmic factors) [6,59,62,67,77], and for this class of MPC algorithms, our compilation also introduces very little asymptotical communication overhead.

[3] We stress that the algorithm itself can be randomized, we just want its communication patterns to be deterministic and fixed a-priori.

Secure multi-party computation for MPC

We now turn to Scenario 2. In this setting security means that a relatively small corrupt coalition cannot learn anything more beyond the coalition's joint outputs. We now ask the following natural question:

> Can we compile any MPC protocol to a *secure* counterpart (where security is in the above sense), allowing only $O(1)$ blowup in round complexity and security parameter related blowup in the total space[4]?

We answer this question affirmatively assuming that the adversary controls only $\frac{1}{3} - \eta$ fraction of machines for any arbitrarily small constant η. Note that $\frac{1}{3}$ is necessary since the MPC model assumes a point-to-point channel without broadcast, and in this model it is known that secure computation cannot be attained in the presence of $\frac{1}{3}$ or more corruptions [48, 82].

To achieve this result, we need to assume the existence of a common random string and appropriate cryptographic hardness assumptions, including the Learning With Errors (LWE) assumption, enhanced trapdoor permutations, as well as the existence of a Fully Homomorphic Encryption (FHE) scheme with an appropriate notion of *compactness* [55, 58]. It is well-known that such compact FHE schemes are implied by a suitable circularly secure variant of the LWE assumption [58], although our compiler can work in general given any such compact FHE scheme (not necessarily based on LWE). Our result is summarized in the following theorem:

▶ **Theorem 2** (Secure computation for MPC). *Assume the existence of a common random string, the Learning With Errors (LWE) assumption, enhanced trapdoor permutations, as well as the existence of an FHE scheme with a suitable notion of compactness (see Appendix A.1 for a formal definition of compactness). Suppose that $s = N^{\varepsilon}$ and that m is upper bounded by a fixed polynomial in N. Let κ denote a security parameter, and assume that $s \geq \kappa$. Given any MPC algorithm Π that completes in R rounds where each of the m machines has s local space, there is an MPC algorithm $\widetilde{\Pi}$ that securely realizes the same function computed by Π in the presence of an adversary that statically corrupts at most $\frac{1}{3} - \eta$ fraction of the machines for an arbitrarily small constant η. Moreover, $\widetilde{\Pi}$ completes in $O(R)$ rounds, consumes at most $O(s) \cdot \mathsf{poly}(\kappa)$ space per-machine, and incurs $O(m \cdot s) \cdot \mathsf{poly}(\kappa)$ total communication per round.*

Now, one interesting question is whether the cryptographic assumptions we rely on in the above theorem can be avoided. We show that if one can indeed achieve the same result with *statistical* security, then it would imply (at least partial) solutions to long-standing open questions in the cryptography literature. Specifically, in Appendix B, we show that if we could construct such a compiler, it would immediately imply a constant-round Secure Multi-Party Computation protocol for a broad class of circuits that can be computed in small space, achieving *total* communication complexity that is (significantly) sublinear in the circuit size, regardless of the number of parties. As noted in numerous works [26, 39, 44, 45], the existence of such constant-round, sublinear-communication multi-party computation (for circuits) with *statistical* security has been a long-standing open problem, even for special (but nonetheless broad) classes of circuits.

[4] Since many well-known MPC algorithms [2, 4–6, 10–13, 15–17, 19, 20, 29, 36, 40, 41, 46, 47, 52, 60, 61, 63, 80, 81, 83, 90, 93, 97, 107] incur only constant to sub-logarithmic rounds, we would like to preserve the round complexity tightly; and thus we do not allow a security parameter related blowup for round complexity.

Interestingly, we note that barring strong assumptions such as Indistinguishable Obfuscation [53], the only known approach to construct constant-round, sublinear-communication *multi*-party computation for circuits of unbounded polynomial size is also through compact FHE [55, 58]. We stress, however, that even with a compact FHE scheme, constructing our "MPC to SMPC-for-MPC" compiler is non-trivial and require the careful combination of several techniques.

2 Technical Roadmap

We now present a succinct and informal technical roadmap to capture our main ideas and new techniques.

Recall that in the MPC model of computation, there are m machines each with s local space. All machines will jointly compute a function over a large input containing N words. We assume that $s = N^{\varepsilon}$ for some constant $\varepsilon \in (0, 1)$, and that $m \in [N^{1-\varepsilon}, \mathsf{poly}(N)]$. Note that although our results will hold as long as m is upper bounded by some polynomial function in N, in known MPC algorithms typically we desire that $m \cdot s$ is not too much greater than N. At the beginning of the first round, every machine receives an input whose size is bounded by s. In every other round, each machine begins by receiving incoming messages from the network, and it is guaranteed that no more than s words will be received such that the machine can write them down in its local memory – henceforth this is referred to as the *s-receiver-constraint*. After receiving the inputs or the network messages, all machines perform local computation, and then send messages to other machines. These messages will then be received at the beginning of the next round.

As explained earlier, in the algorithms literature on MPC, the primary metric of performance is the algorithm's *round complexity* [2, 4–6, 10–13, 15–17, 19, 20, 29, 36, 40, 41, 46, 47, 52, 60, 61, 63, 80, 81, 83, 90, 93, 97, 107].

2.1 Achieving Communication Obliviousness: Oblivious Routing

Many known MPC algorithms are not communication oblivious, and thus the communication patterns of these algorithms can leak information about the secret inputs. For example, many graph algorithms for MPC have communication patterns that will leak partial information about the structure and properties of the graph, such as the degrees of nodes or the connectivity between vertices [6, 40, 62, 81, 93].

Our goal is to compile an MPC algorithm to a communication-oblivious counterpart while preserving its round and space complexity. To achieve this, we will compile each communication round of the original MPC to a constant-round, oblivious protocol that accomplishes the same message routing. Interestingly, the compiled protocol will respect communication-obliviousness in a very strong sense: *its communication patterns are deterministic and independent of the input.*

Sender and receiver constraints

In the MPC model of computation [6, 40, 93], we typically have that in each round, each machine sends at most s words and receives at most s words – henceforth these are referred to as the s-sender-constraint and the s-receiver-constraint respectively. Note that if a sender wants to send the same word to two machines, it is counted twice.

Oblivious routing

Oblivious routing basically aims to obliviously realize the message routing as long as both the s-sender- and s-receiver-constraints are satisfied.

More specifically, suppose that each machine receives at most s send-instructions as input, where each send-instruction contains an outgoing word to be sent and a destination machine for the outgoing word. The joint inputs of all machines guarantee that each machine will receive no more than s words. How can we design a constant-round, communication-oblivious protocol that routes the outgoing words to the appropriate destinations?

▶ Remark 3. It seems that some MPC works in the algorithms literature respect only the s-receiver constraint but not the s-sender constraint. We think most likely, the folklore understanding is that as long as we assume the s-receiver constraint, whether or not there is an s-sender constraint do not really affect the expressive power of the computation model. For completeness, in Appendix C, we describe a round- and space-preserving transformation that compiles any MPC protocol that satisfies only the s-receiver-constraint to one that satisfies both $O(s)$-receiver- and $O(s)$-sender-constraints. This means that all of our results would be applicable to MPC algorithms that satisfy only the s-receiver-constraint but not the s-sender-constraint.

Background

Our approach is partly inspired by algorithmic techniques from the recent Oblivious RAM and oblivious sorting line of work [7, 33, 49, 96, 101]. Specifically, these works propose a RAM algorithm with a fixed memory access pattern that routes elements to random buckets and succeeds with $1 - \exp(-\Omega(Z))$ probability where Z denotes each bucket's capacity (and assuming that the total number of elements is polynomially bounded). To accomplish this, imagine that initially all N elements are divided into $2N/Z$ buckets each of capacity Z such that each bucket is at most half-loaded. Every element is assigned a random label declaring which bucket it wants to go to. Now, these prior works rely on a *logarithmic-depth*, butterfly network of buckets and move the elements along this butterfly network based on their respective labels, until eventually every element falls into its desired bucket – this is guaranteed to succeed with $1 - \exp(-\Omega(Z))$ probability where a failure can only occur if in the middle some bucket in the butterfly network exceeds its capacity – henceforth this is said to be an overflow event.

In summary, the insight we can gain from this elegant construction is the following: roughly speaking, a butterfly network of super-logarithmically sized buckets can obliviously route elements to their desired buckets in the final layer with a deterministic communication pattern (determined by interconnections in the butterfly network); but to ensure correctness, i.e., to ensure that overflow does not happen except with negligible probability, the elements should have random destination labels to achieve good load-balancing properties.

A first attempt

Our idea is to use such a butterfly-network to route the words to their destinations – specifically, one can imagine that each bucket is relatively small such that every machine holds $\Theta(\frac{s}{Z})$ of the resulting buckets; and moreover, the buckets are numbered $1, 2, \ldots, O(m \cdot s/Z)$ respectively. For convenience, we will use the term "element" to mean an outgoing word to be sent. Since every sender already knows the destination machines for each of its input elements, it can basically assign the element to a random bucket within the destination

machine. Henceforth, by "destination label", we mean the index of the destination bucket (as opposed to the destination machine). We are, however, faced with two challenges which prevent us from adopting the known approach in its current form:

- *Load balancing challenge:* First, in our setting, the destination labels of the input elements are not completely random; and thus the load-balancing properties that hold in earlier works [7, 33, 49] for randomly assigned labels no longer hold in our case;
- *Round complexity challenge:* Second, the natural way to adopt the butterfly network is to assign to each machine an appropriate subset of $\Theta(\frac{s}{Z})$ buckets in each layer of the network. However, if $\frac{s}{Z} = \Theta(1)$, then we will incur logarithmic number $\Omega(\log m)$ of rounds (corresponding exactly to the depth of the butterfly network). Later, we shall set Z to be small enough (e.g., $Z = O(\sqrt{s})$ in size) to reduce the number of rounds.

Overcoming the load balancing challenge

To overcome the load balancing challenge, our idea is to run this butterfly network twice: the first time we use it to route every element a *random* destination bucket just like in the earlier works; and the second time we use it to route every element to a random destination bucket within the destination machine they originally wanted to go to. At the end of the second phase, every element will be routed to the machine it wants to go to.

For the first phase, we can rely on the same load balancing arguments as in the previous works [7, 33, 49] to prove that overflow events happen only with negligible probability. For the second phase, we will prove a new stochastic bound showing that the same load-balancing properties hold with a different starting condition (see Section 4.4): *i)* the starting condition must satisfy the s-receiver-constraint; and *ii)* initially the elements are assigned to random input buckets, which is guaranteed by phase 1.

It remains to overcome the round complexity challenge which we explain below.

Overcoming the round complexity challenge

The earlier works rely on a 2-way butterfly network where in each layer i, a local 2-way routing decision is made for every element based on the i-th bit of its destination label. In our new construction, we will compress r layers of work in the original butterfly to a single round, exploiting the fact that each machine local space to store roughly 2^r buckets, where $2^r = \Theta(\frac{s}{Z})$.[5] In this way our new approach requires $O((\log \frac{N}{Z})/r) = O(1/\varepsilon)$ rounds for $Z = O(\sqrt{s})$ and $s = N^{\varepsilon}$. Effectively, in each round i, each machine would be looking at the i-th r-bit-group of an element's label to make a 2^r-way routing decision for the element.

To make this idea work, the crux is to show that at the end of every round (corresponding to r layers in the old 2-way butterfly), there is a communication-efficient way for the machines to exchange messages and rearrange their buckets such that the interconnections in the graph are "localized" in the next round too. In this way, within each round, each machine can perform 2^r-way routing on its local elements, simulating r layers of the old 2-way butterfly, without communicating with any other machine. Fortunately, we can accomplish this by exploiting the structure of the butterfly network. We defer the algorithmic details and proofs to Section 4.

[5] Our techniques remotely reminiscient of the line of work on external-memory ORAM constructions with large, N^{ϵ} CPU private cache [33, 68, 69, 96, 101] – however, all previous works consider a *sequential* setting.

2.2 SMPC for MPC

2.2.1 Informal Problem Statement

We now turn our attention to Scenario 2 where the adversary controls not just the intermediate network routers but also a subset of the machines involved in large-scale computation. As before, given a computation task f that has an efficient but insecure MPC algorithm, we now would like to *securely* realize f also using the MPC model while preserving its efficiency. Here, our security goal is to guarantee that the adversary learns nothing more than what is already implied by the joint outputs of corrupt machines. Such a notion of security can be formally defined using a standard simulation-based paradigm [30], requiring that the real-world protocol must "securely emulate" an ideal-world protocol where all machines simply forward their inputs to a trusted ideal functionality who performs the desired computation task and forwards the outputs to each machine. Intuitively, we require that for any real-world attack that can be performed by a polynomially bounded adversary, there is an ideal-world adversary that can essentially implement the same attack in the ideal world. We refer the reader to Section 5 for a formal definition. Note that our definition follows the same style as the Universal Composition framework [30]. Henceforth, a secure multi-party computation protocol satisfying the aforementioned security notion is said to be an "SMPC-for-MPC" protocol.

Before we describe how to solve this problem, we need to clarify a few points about the model of execution as we marry SMPC and MPC. Since we now have corrupt machines and corrupt machines are allowed to send arbitrary messages to any machine, we can no longer guarantee that each honest machine receive at most s words. Instead, we require that at the end of the every round $r-1$, every machine can write down in its local memory a receiving schedule for round r of the form $\{(f_j, w_j)\}_j$, where $f_j \in [m]$ denotes the index of a sender to anticipate a message from in round r and w_j denotes the number of words expected from f_j. In this way, an honest machine will only save the first w_j words received from every anticipated sender f_j contained in this receiving schedule; all other received messages are discarded immediately.

▶ Remark 4. Recall that since we showed how to compile any MPC protocol to one that has a fixed communication schedule while asymptotically preserving round and space complexity (Section 2.1), requiring that receivers be able to anticipate their receiving schedule does not reduce the expressive power of the underlying computational model. However, somewhat more subtlely, we do not insist that the communication patterns be deterministic (i.e., fully fixed a-priori) for our SMPC-for-MPC protocols in order not to rule out interesting SMPC-for-MPC protocols – it suffices for an honest machine to anticipate its receiving schedule of some round right before the round starts.

2.2.2 MPC to "SMPC-for-MPC" Compiler

Recall that we would like the compiled SMPC-for-MPC protocol to tightly preserve the original MPC program's round complexity, and for per-machine space we only allow a security parameter related blowup. Although in the cryptography literature, secure multi-party computation (SMPC) techniques have been explored for various models of computation including circuits [21, 55, 65, 105, 106], RAM [1, 54, 56, 57, 71, 85, 86, 104], and PRAM [24, 25, 32, 34, 35, 37, 38, 87, 92], none of the existing approaches can satisfy our efficiency requirements due to a discrepancy in both model and metric of performance.

First, any approach whose round complexity depends at least on the depth of the circuit [9, 22, 65, 78] or on the parallel runtime of a PRAM [24] can be immediately ruled out, since MPC protocols can have high circuit/PRAM depth (e.g., if each machine's local

computation is of a sequential nature). We thus turn our attention to known approaches that achieve small round complexity. For example, a line of works [18, 42, 84] showed a technique where multiple machines jointly garble a circuit/PRAM in constant number of rounds, then each individual machine simply evaluates the garbled circuit/PRAM locally. Although such a protocol would indeed complete in a small number of rounds, the garbled circuit/PRAM's size would be at least linear in the total computation time of all parties (unless we make strong assumptions such as Indistinguishable Obfuscation [53]). This means that the per-machine space blowup will be proportional to the number of machines m. Similarly, other constant-round approaches, including those relying on Threshold Fully Homomorphic Encryption [8] or Multi-Key Fully Homomorphic Encryption [14, 28, 72, 75, 91], also do not directly work due to a linear-in-m blowup in the per-machine space, e.g., due to the need to store all machines' encrypted inputs (although later in our construction we will indeed leverage FHE-based MPC techniques as a building block, although this MPC will only be run among small committees).

Our approach

Henceforth we assume that the adversary controls only $\frac{1}{3} - \eta$ fraction of the machines where η may be an arbitrarily small constant. As explained earlier, this is nearly optimal tolerance in a point-to-point network since an honest fraction of at least $\frac{2}{3}$ is necessary for realizing broadcast in a point-to-point network. Without loss of generality, we may also assume that the original MPC has already been compiled to a communication-oblivious counterpart whose communication patterns are deterministic and input independent.

Our idea is to directly emulate the original (communication-oblivious) MPC round by round, by having a randomly elected small committee emulate each machine's actions in the original MPC. Henceforth the committee acting on behalf of machine i in the original MPC is called the i-th committee. As long as the committee size is polylogarithmic in the security parameter, except with negligible probability each committee must have at least a $\frac{2}{3}$-fraction of honest machines. Therefore, we may employ an SMPC that tolerates less than $\frac{1}{3}$ corruption (and satisfies suitable efficiency requirements to be described below) to securely emulate the following actions – note that in all cases this SMPC protocol will be executed among a small set of polylogarithmically many machines; and thus we call this SMPC building block CommitteeSMPC:

1. *Share input*: initially each machine i secret shares its input with the i-th committee; the secret sharing scheme must be *robust* such that correct reconstruction can be achieved in polynomial time even when corrupt machines provide false shares. Note that to achieve this we may run the aforementioned CommitteeSMPC protocol among machine i and the i-th committee;
2. *Local compute*: in every round's computation step, the i-th committee employ CommitteeSMPC to jointly evaluate what machine i's is supposed to locally compute in the original MPC, and the result is again robustly secret-shared among the committee;
3. *Communicate*: whenever machine i wants to send a message to machine j in the original MPC, now the i-th committee and the j-th committee employ CommitteeSMPC to accomplish this. At the end of this small protocol, every member of the j-th committee will obtain a *fresh* robust secret share of the message.

Instantiating CommitteeSMPC with suitable efficiency requirements

For our compilation to satisfy the stated efficiency goals, the CommitteeSMPC employed must meet certain efficiency requirements:

1. *Constant round:* the protocol must complete in constant number of rounds; and
2. *Weakly space efficient:* the space consumed by each machine in the CommitteeSMPC protocol asypmtotically matches the RAM-space-complexity of the function being evaluated, allowing only a security-parameter-related blowup[6]. In this paper, we assume that the RAM-space-complexity accounts for the space for writing down the RAM program's description and all inputs and outputs.

▶ Remark 5. Note that since the RAM-space-complexity accounts for writing down all machines' inputs and outputs, by definition the RAM-space-complexity of the function being evaluated incur a linear blowup in the number of machines. However, since CommitteeSMPC will only be run among at most 2 committees, the number of machines participating is small in our case. In fact, by the weak space efficiency condition, each machine's space complexity may sometimes need to be sublinear in the circuit size, runtime, or even depth of the function being evaluated (depending on the function being evaluated).

The only known approach for achieving these guarantees simultaneously is through Threshold Fully Homomorphic Encryption (TFHE) [8] or Multi-Key Threshold Fully Homomorphic Encryption (MTFHE) [14, 72, 75] with a suitable notion of *compactness* (to be explained shortly after). Roughly speaking, to perform an SMPC, the following takes place where the encryption scheme employed is (M)TFHE:

(a) possibly after a setup phase, each machine encrypts their local input using the (M)TFHE scheme and computes a zero-knowledge proof attesting to the well-formedness of the ciphertext; now the machine broadcasts the ciphertext as well as the proof;
(b) now each machine homomorphically evaluates an encryption of all machines' outputs;
(c) for every machine involved, compute the partial decryption share for that machine's encrypted output, along with a zero-knowledge proof attesting to the correctness of decryption; now send the partial decryption share and the proof to the corresponding machine.
(d) a machine finally reconstructs the output after collecting enough decryption shares.

If the (M)TFHE scheme employed is *compact* in the sense that the public key, secret key, and ciphertext sizes depend only on the security parameter κ but not the size or the depth of the circuit being evaluated, then we can show that in the above steps each machine needs only $O(m' \cdot s) \cdot \mathsf{poly}(\kappa)$ space where m' denotes the number of machines involved in the CommitteeSMPC protocol and s denotes the RAM-space-complexity of the function being evaluated. Specifically, recall that any RAM machine with space complexity s can be converted to a layered circuit with width s (and moreover the circuit can be generated and written down layer by layer consuming space proportional to the RAM's next-instruction circuit size). Thus in the above, Step (b) can be accomplished using $O(m' \cdot s) \cdot \mathsf{poly}(\kappa)$ space; and it is easy to verify that all other steps indeed consume at most $O(m' \cdot s) \cdot \mathsf{poly}(\kappa)$ too.

In existing (M)TFHE schemes [8, 14, 72], however, the key and ciphertext sizes are dependent on the depth of the circuit being evaluated and thus they do not satisfy the aforementioned compactness requirement. To make these schemes compact, we need to rely on the bootstrapping technique described in the original Gentry work on FHE [55]; and it is

[6] We call this notion weakly space efficient, since one can imagine a stronger notion requiring that the *total* space consumed by *all* parties asyptotically matches the RAM-space-complexity of the function being evaluted.

well-known that to get provable security with bootstrapping, we need to assume that the (M)TFHE scheme employed satisfies *circular security* – informally speaking, ciphertexts must nonetheless remain semantically secure even when we use the encryption scheme to encrypt the secret decryption key.

Since there are relatively few known (M)TFHE constructions [14,72], rather than assuming that the existing constructions are circularly secure, we would like to further hedge our bet. Later in our technical sections, we further relax the assumption and base our scheme instead on LWE and the existence of *any* compact FHE. Note that compact FHE is known to exist assuming circularly secure variants of LWE; but for our purpose, we can work with any compact FHE scheme including ones that depend on different algebraic assumptions.

2.3 Related Work

As mentioned earlier, the cryptography literature has extensively considered secure computation on an parallel architecture but most existing works focus on the PRAM model [3, 24, 25, 32, 34, 35, 37, 38, 87, 92]. Since most real-world large-scale parallel computation is now done on an MPC architecture, we hope that our work will bring the MPC computation model (which has been extensively studied in the algorithms literature) to the attention of the cryptography community. Besides the PRAM model, Parter and Yogev have considered secure computation on graphs in the so-called CONGEST model of computation [94, 95].

3 Preliminaries

3.1 Massively Parallel Computation Model

We now describe the *Massively Parallel Computation* (MPC) model. Let N be the input size in words where each word consists of $w = \Omega(\log N)$ bits, and $\varepsilon \in (0, 1)$ be a constant. The MPC model consists of m parallel machines, where $m \in [N^{1-\varepsilon}, \mathsf{poly}(N)]$ and each machine has a local space of $s = N^{\varepsilon}$ words. Hence, the total space of all machines is $m \cdot s \geq N$ words. 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 [2, 5, 79, 83] assume that $m \cdot s = \widetilde{O}(N)$, or $m \cdot s = O(N^{1+\theta})$ for some small constant $\theta \in (0, 1)$. Henceforth the m machines are numbered $1, 2, \ldots, m$ respectively. The m machines are pairwise connected and every machine can send messages to every other machine.

In this paper we are interested in *protocols* (also called *algorithms*) in the MPC model. In this model, the computation proceeds in *rounds*. At the beginning of each round, if this is the first round then each machine receives N/m words as input; else each machine receives incoming messages from the network, and a well-formed MPC algorithm must guarantee that each machine receives at most s words since there is no additional space to store more messages. After receiving the incoming messages or inputs, every machine now performs local computation; and we may assume that the local computation is bounded by $\mathsf{poly}(s)$ and the choice of the polynomial poly is fixed once the parameters s and m are fixed. After completing the local computation, every machine may send messages to some other machines through a pairwise channel, and then all messages are received at the beginning of the next round. When the algorithm terminates, the result of computation is written down jointly by all machines, i.e., by concatenating the outputs of all machines. Every machine's output is also constrained to at most s words. 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.

In the standard literature on MPC, we are most concerned about the *round complexity* of an MPC algorithm. Specifically, this model has been of interest to the algorithms community since numerous tasks that are known to have logarithmic depth lower bounds on the classical Parallel Random-Access Machine (PRAM) model are known to have sublogarithmic- or even constant-round algorithms in the MPC model [40, 67, 79, 93].

Useful notations and conventions

We introduce a couple useful notations and conventions:

- Given an MPC algorithm Π, we use the notation $(y_1, y_2, \ldots, y_m) \leftarrow \Pi(x_1, x_2, \ldots, x_m)$ to denote a possibly randomized execution of the algorithm where each machine i's input is x_i and its output is y_i for $i \in [m]$.
- When we say the input to an MPC algorithm, we mean the concatenation of all machines' inputs. When we say that an input array I is *evenly spread* across the m machines, we mean that every machine but the last one obtains $\lfloor |I|/m \rfloor$ elements and the last machine obtains $|I| \mod m$ elements where $|I|$ denotes the total number of elements in I.
- We use the term *s-receiver-constraint* to refer to the requirement that a well-formed MPC algorithm must ensure that each machine receives no more than s words in each round. One may also consider, symmetrically, an *s-sender-constraint*, that is, in each round every machine can send at most s words (where sending the same word to two machines counts twice). Many MPC algorithms in the literature respect both constraints. However, it seems that some other works in the MPC literature require only the *s-receiver-constraint* but not the *s-sender-constraint*.

  It turns out that the two modeling approaches are equivalent in terms of expressive power as we show in Appendix C. Therefore, in the main body of the paper, we simply assume that both constraints must be respected, but our results also extend to MPC algorithms that respect only the s-receiver-constraint.
- Like in the standard algorithms literature on MPC, we are concerned about *asymptotical* complexity. For this reason, whenever convenient, we shall assume that each machine is allowed $O(s)$ local space rather than s. Similarly, the s-receiver-constraint is sometimes interpreted as each machine receiving no more than $O(s)$ data per-round.
- We assume that the original MPC protocol to be compiled runs in a *fixed* number of rounds. If not, we can always pad its round complexity to be the worst case. This ensures that no information will be leaked through the number of rounds.

3.2 Communication-Oblivious MPC Algorithms

Communication-oblivious MPC algorithms

To compile an MPC algorithm to a secure counterpart, we go through an important stepping stone where we first compile the original MPC algorithm to a communication-oblivious counterpart. As mentioned earlier in Section 1, communication-oblivious MPC algorithms are also interesting in their own right, e.g., for MPC clusters where the end points are secured with secure processors such as Intel SGX, such that the adversary can observe only the communication patterns.

Intuitively, an MPC algorithm is said to be communication-oblivious, iff an adversary who can observe the communication patterns of the machines learn nothing about the secret input. When we execute an MPC algorithm on some input I, the *communication pattern* incurred in the execution is the concatenation of the following:

1. For each round r, a matrix $P_r \in [s]^{m \times m}$ where $P_r[i,j] \in [s]$ indicates how many words machine i sends to machine j in round r;
2. An ordered list containing the number of words each machine outputs at the end of the algorithm.

In this paper, we define a very strong notion of communication obliviousness: we say that an MPC algorithm is *communication-oblivious*, iff the communication pattern of the algorithm is deterministic, input independent, and known a-priori. Note that this also implies that the algorithm must run for a deterministic number of rounds.

Defining correctness of MPC algorithms

We will define a notion of δ-correctness for an MPC algorithm. Let $(y_1, y_2, \ldots, y_m) \leftarrow \mathcal{F}(x_1, x_2, \ldots, x_m)$ be a (possibly randomized) ideal functionality which, upon receiving inputs $x_1, x_2, \ldots, x_m$, outputs $y_1, y_2, \ldots, y_m$. Here x_i represents machine i's input and y_i represents machine i's output for $i \in [m]$; without loss of generality we may assume that each x_i contains exactly s words (if not we can always pad it with dummies to exactly s words). We say that an MPC algorithm denoted Π δ-correctly realizes the ideal functionality $\mathcal{F}$ iff for any input $I = (x_1, \ldots, x_m)$ the statistical distance between $\Pi(I)$ and $\mathcal{F}(I)$ is at most δ.

▶ **Definition 6** (δ-oblivious realization of an ideal functionality)**.** *We say that an MPC algorithm Π δ-obliviously realizes an ideal functionality $\mathcal{F}$, iff Π is communication-oblivious and moreover Π δ-correctly realizes $\mathcal{F}$.*

▶ Remark 7. One can alternatively consider a weaker notion for an MPC algorithm Π to "δ-oblivious realize" the ideal functionality $\mathcal{F}$, that is, we require that there exists a simulator Sim, such that for any input I of appropriate length, the following distributions must have statistical distance at most δ:

- $\mathbf{Real}^{\Pi}(1^m, 1^s, I)$: outputs the outcome of the MPC algorithm Π on input I and its communication patterns;
- $\mathbf{Ideal}^{\mathcal{F}}(1^m, 1^s, I)$: outputs $\mathcal{F}(I)$ and $\mathsf{Sim}(1^m, 1^s)$. Notice that the simulator Sim is not given the input I.

It is not hard to see that our notion (i.e., Definition 6) implies this weaker notion. This weakened definition would permit the communication patterns to be randomized and also not necessarily known a-priori. In this paper we focus on the stronger notion since we can achieve even the stronger notion in an efficiency-preserving manner.

4 Oblivious Routing and Communication-Oblivious Compiler

4.1 Problem Definition

Recall that our first goal is to obtain an MPC protocol communication oblivious, and the crux of the problem is to realize an oblivious form of routing. Imagine that in the original MPC protocol, in some round, every machine has a list of at most s words that they want to send, and each outgoing word has an intended destination machine. It is also guaranteed that every machine will receive no more than s words. The question is how to route these messages obliviously such that the communication patterns leak no information.

More formally, the $\mathsf{Routing}$ problem has the following syntax:

- **Input.** Every machine $i \in [m]$ has *at most s send instructions* where each send instruction consists of a word to be sent, and the index of the recipient machine. All machines' send instructions must jointly satisfy the s-receiver-constraint, i.e., every machine receives no more than s words.

- **Output.** Every machine outputs the list of words it is supposed to receive as uniquely defined by the joint input (i.e., all machines' send instructions); and moreover, these received words are sorted based on a lexicographically increasing order. If fewer than s words are received, the machine pads the output array with dummies to a length of exactly s.

The above abstraction defines a most natural ideal functionality $\mathcal{F}_{\text{Routing}}$ which implements the routing task correctly.

In the remainder of this section, we will devise an oblivious MPC protocol that accomplishes routing. In the process we will need two intermediate building blocks called "Bucket Route" and "Random Bucket Assignment" respectively.

Notational convention: a global Overflow indicator

Throughout this section, we shall assume that each machine maintains a global variable denoted Overflow. Initially the Overflow bit is set to 0. During the algorithm's execution, a machine may set the Overflow bit to 1. If at the end of the algorithm, all machines' Overflow indicators remain unset, we say that the algorithm is successful; otherwise the algorithm is said to have failed.

4.2 Building Block: Bucket Route

4.2.1 Syntax

The goal is to classify elements into buckets based on each element's label indicating its desired destination bucket. The algorithm is not always guaranteed to succeed; however, should it succeed, the final assignment should be correct. Recall that an algorithm is said to be successful if no machine has set their Overflow indicator by the end of the algorithm. We consider both the input and output configuration as a list of buckets spread evenly across the machines, where each bucket contains either real or dummy elements.

More formally, a $\mathsf{BucketRoute}^Z$ algorithm, parametrized by a bucket size Z, satisfies the following syntax – recall that there are in total m machines each of which has local space $O(s)$; without loss of generality, we may assume that m and s are both powers of 2:

- **Input.** In the beginning, each machine stores 2^r number of buckets each of capacity Z where $2^r \cdot Z = 2s$. Each bucket contains Z elements, some of which are real and others are dummy. Each real element carries an ℓ-bit label where $2^\ell = m \cdot 2^r$ – henceforth if a real element's label is $k \in \{0,1\}^\ell$, we say that the element wants to go to the k-th bucket out of a total of $2^\ell = m \cdot 2^r$ buckets.
- **Output.** The machines jointly output a list of $2^\ell = m \cdot 2^r$ buckets, each of capacity Z. We require that if the algorithm is successful (i.e., no machine's Overflow indicator is set), then every bucket must contain all the real elements wanting to go there plus an appropriate number of dummy elements.

4.2.2 Protocol

We now describe a $\mathsf{BucketRoute}^Z$ algorithm that tries to move every real element to the desired destination bucket but will possibly fail in the middle due to overflow events. Later in Section 4.4, we will show that if the input configuration satisfies certain nice conditions, then the overflow probability can be made negligibly small. Roughly speaking, by "nice conditions", we want that all input buckets are at most half-full; and moreover the input elements' labels are randomly chosen. In this section, we first focus on presenting the algorithm and we will worry about the probability analysis in Section 4.4.

Algorithm 1 Bucket Route.

Input: Let $2^r \cdot Z = 2s$ and let $\ell = \log_2 m + r$. The input consists of 2^ℓ buckets denoted $\mathcal{L} := \{\mathsf{B}_i : i \in [2^\ell]\}$ each with capacity Z, where each real element in a bucket contains an ℓ-bit label. There are $m = 2^{\ell - r}$ parallel machines, each of which has enough memory to store and process 2^r buckets.

1: **procedure** $\mathsf{BucketRoute}^Z(\mathcal{L} := \{\mathsf{B}_i : i \in [2^\ell]\})$ ▷ The input is a list of 2^ℓ buckets.
2: Each bucket in $\mathcal{L}$ receives an empty bit string as its label.
3: **for** $\lfloor \frac{\ell}{r} \rfloor$ sequential iterations **do**
4: Partition the buckets in $\mathcal{L}$ into m groups of equal size 2^r, where the buckets in each group has the same label; a machine is assigned to each group.

▷ *Step 4 requires the machines to exchange their buckets according to a predetermined fashion.*

5: **for** each of m machines in parallel **do**
6: Every machine calls 2^r-way $\mathsf{LocalClassify}^{r,Z}$ on its group of buckets to produce its modified group of buckets (whose labels have lengths increased by r).
7: Update the list $\mathcal{L}$ to be the union of all machines' modified groups of buckets.
8: **if** $t := \ell \mod r \neq 0$ **then**
9: In the last iteration, each machine receives 2^{r-t} sub-groups of buckets, where each sub-group contains 2^t buckets with the same label.
10: **for** each of m machines in parallel **do**
11: Every machine calls 2^t-way $\mathsf{LocalClassify}^{t,Z}$ on every of its sub-groups of buckets.
12: Let $\mathcal{L}$ be the union of all machines' updated buckets.
13: **return** the list $\mathcal{L}$ of 2^r buckets, each of which receives a unique label in $\{0,1\}^r$. Recall that if any instance of multi-way $\mathsf{LocalClassify}$ encounters $\mathsf{Overflow}$, the failure-indicator $\mathsf{Overflow}$ will be set to 1 but the algorithm will continue after truncating the excessive elements in the overflowing bucket(s).

At a very high level, our $\mathsf{BucketRoute}^Z$ algorithm will proceed in iterations such that at the end of the i-th iteration, elements will be sorted based on the first $i \cdot r$ bits of their respective labels. During each iteration i, every machine will obtain a list of buckets containing elements whose labels all share the same $((i-1) \cdot r)$-bit prefix, and the machine will locally further classify these elements into buckets based on the next r bits of their respective labels. This subroutine is henceforth called $\mathsf{LocalClassify}$ which is described more formally below. We will then describe the full $\mathsf{BucketRoute}^Z$ algorithm after specifying this subroutine.

A multi-way LocalClassify subroutine

We introduce a subroutine called a 2^t-way $\mathsf{LocalClassify}^{t,Z}$. The 2^t-way $\mathsf{LocalClassify}^{t,Z}$ subroutine is always performed by a single machine locally which operates on a list of 2^t buckets:

- Imagine that a machine has as input a list of 2^t buckets each of capacity Z where $2^t \cdot Z \leq 2s$. All of the 2^t buckets must have the same bucket-label $x \in \{0,1\}^{|x|}$, where $|x|$ is the bit-length of x. Moreover, every real element contained in the buckets has an ℓ-bit label of the form $x||y$, i.e., the element's label must be prefixed by x, followed by a suffix denoted y (that has length at least r). It is guaranteed that $|x| + r \leq \ell$.
- Now, the machine locally rearranges the input to form 2^t output buckets as follows: for each $i \in \{0,1\}^t$, the i-th output bucket has the bucket-label $x||i$, and contains all the real elements from the input whose label is prefixed with $x||i$, padded with dummies to a capacity of Z. Moreover, in each output bucket the real elements must appear before the dummies.
  If the above rearrange fails due to the overflow of some output bucket, then set the indicator Overflow to 1; moreover, truncate the bucket to a load of Z and continue with the algorithm. Note that if Overflow is set, then some elements can be lost due to the truncation.

The BucketRoute algorithm

We describe the $\mathsf{BucketRoute}^Z$ algorithm in Algorithm 1 which calls LocalClassify as a subroutine. Assuming that no Overflow is encountered, then the algorithm proceeds in the following way: first, using the 2^r-way LocalClassify subroutine, all machines rearrange their local 2^r buckets of elements based on the first r bits of the elements' labels, such that all elements whose labels have the same r-bit prefix fall into the same bucket, and this common r-bit prefix also become the bucket's label. At the end of this iteration, for each r-bit bucket label, there will be $2^{\ell-r}$ buckets with the same bucket label. We will then assign $2^{\ell-2r}$ machines to work on buckets sharing each r-bit bucket label, and each machine now locally calls LocalClassify to further classify the elements based on the next r-bits of their input labels; and this goes on. In general, after the end of the i-th iteration, buckets will acquire labels of length $i \cdot r$ bits, and there will be $2^{\ell-i\cdot r}$ buckets sharing each unique $(i \cdot r)$-bit label; we now assign all buckets with the same label to $2^{\ell-(i+1)\cdot r}$ machines which further classifies the elements based on the next r bits in the input labels. The algorithm will have ended by iteration i if $i \cdot r \geq \ell$, i.e., after $O(\ell/r)$ iterations – at the end, all buckets across all machines will have a unique ℓ-bit label. Algorithm 1 formalizes the above idea and in particular, treats the last iteration more formally for the case when ℓ is not divisible by r.

▶ **Fact 8.** *Using the parameters in Algorithm 1, the number of rounds is* $O(\frac{\ell}{r}) = O(\frac{\log m}{r})$.

▶ **Fact 9** (Communication pattern)**.** *The communication pattern of Algorithm 1 is deterministic, and depends only on the parameters m, ℓ, r and Z. Moreover, in every round, the total communication is upper bounded by $O(m \cdot s)$.*

Proof. Step 4 of Algorithm 1 is where the communication happens, and it is not hard to check that this fact holds. ◀

4.3 Building Block: Oblivious Random Bucket Assignment

Based on the $\mathsf{BucketRoute}^Z$ primitive described above, we realize another intermediate building block called "random bucket assignment". The problem, henceforth denoted $\mathsf{RandBucket}^Z$, is parametrized by an appropriate bucket size Z and is described below:

- **Input.** The input is an array of $m \cdot s$ elements spread evenly across the machines where each machine receives exactly s elements; each element can either be real or dummy. We assume that each element can be stored in $O(1)$ words.

- **Output.** Each machine receives $2^r = 2s/Z$ output buckets each of capacity Z. The contents of the buckets are determined below:
  - Every real element in the input array is assigned to a random bucket out of the $2^r \cdot m$ buckets;
  - If a bucket receives more than Z real elements, choose the Z lexicographically smallest elements to populate the bucket (and the remaining elements are lost);
  - Else if a bucket receives Z or fewer real elements, then the bucket should contain all of these elements, in a lexicographically increasing order, and padded at the end with an appropriate number of dummies to a capacity of Z.

Note that the above abstraction also defines the most natural ideal functionality $\mathcal{F}^Z_{\text{RandBucket}}$ which correctly implements the above task.

Protocol

Using $\mathsf{BucketRoute}^Z$, we can obliviously realize random bucket assignment using the following simple algorithm where we assume that m and s are powers of 2 without loss of generality (if not, we can pad them to the nearest power of 2):

1. Choose r, ℓ based on Z, m, and s, such that $2^r = 2s/Z$ and $2^\ell = m \cdot 2^r$. Henceforth 2^r denotes the number of buckets on each machine, and 2^ℓ denotes the total number of buckets across all machines.
2. Each machine places $Z/2$ elements from the input array to each of its 2^r buckets, and pads each bucket with an additional $Z/2$ dummies to a capacity of Z – note that each bucket is at most half full.
3. Every machine assigns a random ℓ-bit label to each real element in its buckets, indicating which bucket the element wants to go to.
4. Call $\mathsf{BucketRoute}^Z$ (Algorithm 1) to produce the list of output buckets, and for each resulting bucket, sort the elements in it based on a lexicographically increasing order, placing all dummies at the end.

Recall that in the $\mathsf{BucketRoute}^Z$, a machine may encounter a bucket overflow exception which will cause the $\mathsf{Overflow}$ indicator to be set. Below we bound the probability of seeing $\mathsf{Overflow}$ using the fact that the initial labels are randomly chosen and that the initial buckets are at most half full – this allows us to prove good load-balancing properties. We use the standard Chernoff Bound to analyze overflow probability.

▶ **Fact 10** (Chernoff Bound). *Suppose X is a sum of independent $\{0,1\}$-random variables. Then, for any $\beta \geq 2$, $\Pr[X \geq \beta E[X]] \leq \exp(-\frac{\beta E[X]}{6})$.*

▶ **Lemma 11** (Overflow probability for random bucket assignment algorithm). *In the above algorithm, the probability that some machine sets the $\mathsf{Overflow}$ indicator is at most $O(\frac{\ell}{r}) \cdot 2^\ell \cdot e^{-\frac{Z}{6}}$.*

Proof. Observe that there are $O(\frac{\ell}{r})$ iterations in Algorithm 1. We fix some iteration and some bucket B, and analyze the overflow event of that bucket after that iteration.

Observe that if we assume that all buckets from previous iterations have infinite capacity, then no element will be lost in previous iterations. Hence, the resulting number of elements ending up at bucket B after this iteration in this alternate world will stochastically dominate that in the actual world. The property of no overflow in previous iterations is used to ensure that different elements do not influence one another so that we can apply Chernoff Bound. By stochastic dominance, it suffices to analyze the overflow probability of bucket B after this iteration under this additional assumption.

Suppose after this iteration, this bucket has received a label with k bits. This means that the elements in this bucket could only have come from 2^k possible input buckets, each of which contains at most $\frac{Z}{2}$ elements.

Observe that each such element receives a random ℓ-bit label, whose k-bit prefix coincides with this bucket's label with probability $\frac{1}{2^k}$. Hence, the expected number of elements entering this bucket after this iteration is at most $\frac{Z}{2}$.

Therefore, using Chernoff Bound (Fact 10), after each iteration, each bucket overflows with probability at most $e^{-\frac{Z}{6}}$.

Finally, the union bound over all iterations and all buckets gives the result. ◀

▶ **Lemma 12** (Correctness of random bucket assignment)**.** *The above algorithm, parametrized with Z, δ-correctly realizes $\mathcal{F}^Z_{\text{RandBucket}}$ where $\delta = O(\frac{\ell}{r}) \cdot 2^\ell \cdot e^{-\frac{Z}{6}}$ (note that the parameters ℓ and r are determined once Z is determined).*

Proof. Follows from Lemma 11 and the fact for every random string ρ assigning real elements to destination buckets, if ρ does not cause Overflow in the algorithm, then the algorithm's output must correctly match that output by the ideal functionality $\mathcal{F}^Z_{\text{RandBucket}}$ consuming the same randomness ρ. ◀

▶ **Fact 13.** *The communication patterns of the algorithm is deterministic and depends only on the parameters m, ℓ, r and Z.*

Proof. The communication pattern of the algorithm stems from that of $\mathsf{BucketRoute}^Z$ since all other steps are performed locally on each machine and incur no communication. The fact now follows directly from Fact 9. ◀

4.4 Putting it Together: Oblivious Routing

Using the building blocks $\mathsf{BucketRoute}^Z$ and $\mathsf{RandBucket}^Z$, we can realize oblivious routing as follows:

(a) Without loss of generality, we may assume that both m and s are powers of 2 and s is a perfect square. Choose the parameter $Z = 2\sqrt{s}$ and $2^r = 2s/Z$, and $\ell := \log_2 m + r$.

(b) Every machine does the following: let X be an input array containing all the words the machine wants to send (also called outgoing words), padded with dummies to a length of s. Each real outgoing word in the input array has a label of the form $x||\rho$ where $x \in \{0,1\}^{\log_2 m}$ encodes the identifier of the recipient machine, and $\rho \xleftarrow{\$} \{0,1\}^{\ell - \log_2 m}$ is chosen at random and indicates the index of the destination bucket within machine x.

(c) Invoke an oblivious $\mathsf{RandBucket}^Z$ algorithm to assign each outgoing word to a random bucket among a total of 2^ℓ buckets, and recall that each machine stores 2^r of these buckets.

We emphasize that the labels selected earlier in Step (b) are not used as destination labels for the $\mathsf{RandBucket}^Z$ algorithm since the $\mathsf{RandBucket}^Z$ algorithm itself internally assigns a random bucket to each element. However, these labels selected earlier will be used in the next step.

(d) Invoke an instance of $\mathsf{BucketRoute}^Z$ to route each outgoing word to the destination bucket encoded in the label chosen in Step (b). Now, every machine looks at the resulting 2^r buckets it stores and outputs all received (real) words in a lexicographically increasing order, padded with an appropriate number of dummies to a length of s.

▶ **Fact 14.** *In the above routing algorithm, the communication pattern is deterministic and depends only on the parameters Z, m, r, ℓ.*

Proof. The communication pattern of the above algorithm is determined by the communication pattern of the $\mathsf{RandBucket}^Z$ algorithm and the $\mathsf{BucketRoute}^Z$ algorithm. The fact now follows directly from Fact 9 and Fact 13. ◀

▶ **Lemma 15** (Correctness of routing). *Suppose that $\mathsf{RandBucket}^Z$ δ-correctly realizes $\mathcal{F}^Z_{\text{randbucket}}$. Then, the above algorithm δ'-correctly realizes $\mathcal{F}_{\text{Routing}}$ for some $\delta' \leq \delta + O(\frac{\ell}{r}) \cdot 2^\ell \cdot e^{-\frac{Z}{6}}$.*

Proof. We consider a sequence of hybrids.

$\mathbf{Real}^Z(I)$. We will use $\mathbf{Real}^Z(I)$ to denote the outcome of the real-world algorithm.

$\mathbf{Hyb}_1^Z(I)$. We now consider a hybrid execution denoted $\mathbf{Hyb}_1^Z$, which is defined almost the same as the real-world execution, except that we now replace the $\mathsf{RandBucket}^Z$ algorithm with $\mathcal{F}^Z_{\text{randbucket}}$. We define $\mathbf{Hyb}_1^Z(I)$ to output the outcome of this hybrid execution upon the input I.

Since $\mathsf{RandBucket}^Z$ δ-correctly realizes $\mathcal{F}^Z_{\text{randbucket}}$, it holds that for any I, $\mathbf{Hyb}_1^Z(I)$ has at most δ statistical distance from $\mathsf{Real}^Z(I)$.

▶ **Lemma 16.** *For any I, in the execution defined by $\boldsymbol{Hyb}_1^Z(I)$, the probability that some machine sets the $\mathsf{Overflow}$ indicator is bounded by $O(\frac{\ell}{r}) \cdot 2^\ell \cdot e^{-\frac{Z}{6}}$.*

Proof. To prove this, we instead consider an execution $\mathbf{Hyb}_1^\infty$ where the parameters Z, ℓ, and r are chosen as before; however, we do not impose a Z-capacity limit on any of the buckets, including the buckets in $\mathcal{F}_{\text{randbucket}}$ or any bucket in the hybrid execution $\mathbf{Hyb}_1^Z$. It is not hard to that the probability of seeing $\mathsf{Overflow}$ in $\mathbf{Hyb}_1^Z$ is upper bounded by the probability that there is some bucket storing more than Z real elements in $\mathbf{Hyb}_1^\infty$. This can be formally proven through a standard stochastic domination argument.

Therefore, it suffices for us to prove that in $\mathbf{Hyb}_1^\infty$, the probability that some bucket needs to store more than Z real elements is upper bounded by $O(\frac{\ell}{r}) \cdot 2^\ell \cdot e^{-\frac{Z}{6}}$, and below we prove this statement.

The analysis is similar to the proof of Lemma 11. We fix some iteration in Algorithm 1 and some bucket B. Let $X_{i,B}$ denote an indicator random variable indicating the contribution of the i-th outgoing word in the input to the load of the bucket B. Our goal is to prove a tail bound for $\sum_i X_{i,B}$ which denotes bucket B's total load. Note that $X_{i,B}$ depends only on the i-th outgoing word's destination bucket and the initial bucket placement chosen by $\mathcal{F}_{\text{RandBucket}}$ for the i-th outgoing word. Therefore, all of the random variables $\{X_{i,B}\}_i$ are independent which will allow us to apply the standard Chernoff bound. To do so, we will first show that the expected number of real elements the bucket contains after this iteration is at most $\frac{Z}{2}$. Hence, in the execution $\mathbf{Hyb}_1^\infty$, we analyze the expected number of elements for this bucket B after this iteration based on the number k of bits of the label received by this bucket after this iteration. Let $\mathcal{L}$ be the list of buckets obtained after $\mathcal{F}_{\text{randbucket}}$.

1. Suppose $k \leq \log_2 m$. The k-bit label indicate a subset of 2^{m-k} destination machines. Then, only elements going to these 2^{m-k} machines can end up in this bucket. There are totally at most $2^{m-k} \cdot s$ such elements.
   Moreover, only elements from 2^k buckets in $\mathcal{L}$ can end up in bucket B after this iteration. Due to $\mathcal{F}_{\text{randbucket}}$, an element is in one of those 2^k buckets independently with probability $\frac{2^k}{2^\ell}$.
   Hence, the expected number of elements in bucket B after this iteration is at most $2^{m-k} \cdot s \cdot \frac{2^k}{2^\ell} = \frac{s}{2^r} = \frac{Z}{2}$.

2. Suppose $k > \log_2 m$. The k-bit label then identifies $2^{\ell-k}$ buckets within a certain destination machine M.
   Recall that at most s elements are supposed to go to machine M. In order for an element to end up in bucket B after this iteration, it has to satisfy the following independent events:
   a. The element is in one of 2^k buckets in $\mathcal{L}$ whose elements can end up in B after this iteration. Due to $\mathcal{F}_{\text{randbucket}}$, this happens with probability $\frac{2^k}{2^\ell}$.
   b. The element has chosen a destination bucket that is among those $2^{\ell-k}$ buckets whose identity agrees with the k-bit label of the bucket B. This happens with probability $\frac{2^{\ell-k}}{2^r}$.
   Hence, the expected of elements that enter this bucket B after this iteration is at most $s \cdot \frac{2^k}{2^\ell} \cdot \frac{2^{\ell-k}}{2^r} = \frac{s}{2^r} = \frac{Z}{2}$, as required.

Finally, we apply a standard Chernoff bound and then a union bound over all iterations and all buckets just like in Lemma 11, which leads to the lemma statement. ◀

Ideal(I). The experiment $\mathbf{Ideal}(I)$ outputs the result of $\mathcal{F}_{\text{Routing}}$ upon the input I

We now show that for any I, $\mathbf{Ideal}(I)$ and $\mathbf{Hyb}_1^Z$ have statistical distance at most $O(\frac{\ell}{r}) \cdot 2^\ell \cdot e^{-\frac{Z}{6}}$. Note that conditioned on 1) seeing no Overflow in $\mathbf{Hyb}_1^Z$ and 2) that $\mathcal{F}_{\text{randbucket}}$ does not pick bad randomness such that some bucket initially exceeds Z load, then the outcome obtained in $\mathbf{Hyb}_1^Z$ would be the same as the output of $\mathbf{Ideal}(I)$. In Lemma 16, we have shown that the probability of some machine setting the Overflow indicator in $\mathbf{Hyb}_1^Z$ $O(\frac{\ell}{r}) \cdot 2^\ell \cdot e^{-\frac{Z}{6}}$, therefore it suffices to show that the probability that $\mathcal{F}_{\text{randbucket}}$ internally exceeds Z load is upper bounded by $O(\frac{\ell}{r}) \cdot 2^\ell \cdot e^{-\frac{Z}{6}}$. Due to a simple application of the Chernoff bound, we can show that the probability that some fixed bucket exceeds load Z is at most $e^{-\frac{Z}{6}}$. Now, applying a union bound over all 2^ℓ buckets, the conclusion follows.

In summary, applying a standard hybrid argument, we can complete the proof of this lemma. ◀

▶ **Corollary 17.** *The above algorithm δ-obliviously realizes $\mathcal{F}_{\text{Routing}}$ by Definition 6 for $\delta = \exp(-\Omega(\sqrt{s}))$.*

Proof. Straightforward by Fact 14, Lemma 15 and Lemma 12 and the parameters chosen in the algorithm. ◀

4.5 Oblivious Sorting

Given our oblivious routing primitive, one immediate and interesting application is to construct a constant-round MPC algorithm that obliviously realizes sorting.

To achieve this, recall that Goodrich [67] constructed a non-oblivious MPC algorithm that accomplishes sorting in $O(1)$ rounds; and moreover, his algorithm satisfies both the s-receiver-constraint and the s-sender-constraint.

▶ **Lemma 18** (Constant-round sorting, Theorem 3.5 of Goodrich [67]). *Let $m = N^{1-\varepsilon}$ and $s = O(N^\varepsilon)$ for any constant $\varepsilon \in (0, 1)$. Suppose that each item to be sorted can be stored in $O(1)$ words. There exists a deterministic, comparison-based sorting algorithm that can correctly sort N items stored on m machines each with s local space consuming an a-priori fixed constant number of rounds; and moreover, the algorithm satisfies both the s-sender-constraint and the s-receiver-constraint.*

The idea is to apply our Routing algorithm to realize each round of communication in Goodrich's algorithm. In this way, we obtain the following theorem:

▶ **Theorem 19** (Constant-round oblivious sorting on MPC). *Let $m = N^{1-\varepsilon}$ and $s = O(N^{\varepsilon})$ for any constant $\varepsilon \in (0, 1)$, and suppose that each items to be sorted can be represented in $O(1)$ words. There exists an MPC algorithm that obliviously sorts N items stored on s machines in $O(1)$ number of rounds and the result is correct with $1 - \exp(-\Omega(\sqrt{s}))$ probability.*

Proof. Note that Goodrich's algorithm runs for an apriori-fixed constant number of rounds. Thus, if we apply our oblivious Routing algorithm to realize each round of communication in Goodrich's algorithm, the resulting algorithm has a deterministic communication pattern; and moreover, due to Corollary 17, the resulting algorithm correctly sort the input with $1 - \exp(-\Omega(\sqrt{s}))$ probability. ◀

Although our secure multi-party computation compiler later will not directly rely on oblivious sorting as a building block, we state this result explicitly nonetheless since sorting is such an important and fundamental algorithmic building block.

4.6 Communication-Oblivious Compiler

We can now arrive at Theorem 1 which we restate below for the reader's convenience.

▶ **Theorem 20** (Communication-oblivious MPC algorithms: Restatement of Theorem 1). *Suppose that $s = N^{\varepsilon}$ and that m is upper bounded by a fixed polynomial in N. Given any MPC algorithm Π that completes in R rounds where each of the m machines has s local space, there is a communication-oblivious MPC algorithm $\widetilde{\Pi}$ that computes the same function as Π except with $\exp(-\Omega(\sqrt{s}))$ probability, and moreover $\widetilde{\Pi}$ completes in $O(R)$ rounds, and consuming $O(s)$ space on each of the m machines. Furthermore, only $O(m \cdot s)$ amount of data are communicated in each round in an execution of $\widetilde{\Pi}$.*

Proof. We apply the oblivious Routing algorithm developed earlier in this section to realize every communication round of the original MPC protocol Π. The theorem now follows directly from Corollary 17. ◀

5 Secure Multi-Party Computation for Massively Parallel Computing

In this section, we consider how to perform secure computation on a Massively Parallel Computing (MPC) architecture. As before, we consider a set of m machines each of which is s-space-bounded. Without loss of generality, we consider a setting where each machine receives some input denoted $x_1, x_2, \ldots, x_m$ respectively where each input contains at most s words. Now, these machines would like to jointly evaluate a function $(y_1, y_2, \ldots, y_m) \leftarrow f(x_1, x_2, \ldots, x_m)$ such that at the end, the i-th machine obtains the s-bounded output y_i for $i \in [m]$. We would like to ensure that an adversary controlling a relatively small subset of the machines will not learn anything beyond the outputs of the machines in its control.

The question we are concerned about is the following: suppose that there is an efficient, insecure MPC protocol Π to evaluate the function f. Can we now securely evaluate the function f while preserving the efficiency relative to the insecure protocol Π?

5.1 Execution Model: SMPC for MPC

We consider an MPC protocol Π executing on m machines each with maximum space s. To define Secure Multi-Party Computation (SMPC), we augment the protocol execution model used so-far in the paper as follows to capture a polynomial-time adversary who can corrupt a

subset of the machines. Most of the definitions below directly follow the standard approach in the cryptography literature – there is, however, one subtlety that needs to be clarified when we marry SMPC and MPC: since corrupt machines can send arbitrary messages to honest machines, we must now redefine the s-receiver-constraint (see details in the "communication model" paragraph).

- We now parametrize the protocol's execution with a security parameter denoted κ. Therefore we may write Π as $\Pi(1^\kappa, 1^m, 1^s)$, i.e., it is parametrized by the security parameter κ and the MPC framework's parameters 1^m and 1^s.
- A subset of the machines which are said to be *corrupt* are controlled by an adversary $\mathcal{A}(1^\kappa)$. We assume that corruption is static, i.e., $\mathcal{A}$ decides which machines to corrupt before the protocol execution starts. All protocol messages received by corrupt machines are visible to $\mathcal{A}$, and $\mathcal{A}$ fully controls what messages corrupt machines send.

  Machines that are not in $\mathcal{A}$'s control are said to be honest, and honest machines faithfully follow the prescribed protocol.
- All machines' inputs are chosen by some environment denoted $\mathcal{Z}(1^\kappa)$; and at the end of the protocol, all honest machines send their respective output to $\mathcal{Z}$.
- During the execution $\mathcal{A}$ and $\mathcal{Z}$ may communicate freely.

Communication model

The protocol proceeds in rounds and machines communicate with each other through a pairwise point-to-point network. At the beginning of each round, honest machines receive messages from the network; and afterwards they perform computation and send messages. If an honest machine sends a message in round r, then an honest recipient will receive the message at the beginning of the next round. We assume that honest machines communicate through a pairwise *secure channel* such that the adversary can observe who is communicating with whom, the length of each honest message sent, but not the contents of the message – note that since we can realize secure channels from authenticated channels with key exchange and authenticated encryption, assuming secure channels is without loss of generality.

Recall that in this new setting, some machines can be corrupt and corrupt machines send arbitrary, unwanted messages to honest machines. We cannot guarantee that the s-receiver-constraint is respected in the presence of corrupt nodes; instead, we require the following:

> at the end of the previous round, an honest machine must have written down in a designated location in its memory a *receiving schedule* for the next round, i.e., a list of $\{(f_j, w_j)\}_j$ pairs where $f_j \in [m]$ denotes the index of a sender and $w_j \in [s]$ denotes the number of words the machine is expecting to receiver from machine f_j. Additionally, it must hold[7] that $\sum_j w_j \leq s$.

Every sender whose index appears in this receiving schedule is called an anticipated sender. Now, a machine will save only the first w_j words received from each anticipated sender f_j; it will ignore all words received from unanticipated senders; and also ignore all excessive words received from anticipated senders.

[7] Later on, in our compiled protocol, s will actually be substituted with $O(s) \cdot \mathsf{poly}(\kappa)$ where s is the per-machine space complexity of the original MPC to be compiled.

5.2 Security Definition

Security is defined through (computational) observational equivalence of the environment $\mathcal{Z}$ in an ideal-world and a real-world execution. In the real-world execution, machines run the real protocol Π. In the ideal-world execution, the computation task is performed by an ideal functionality $\mathcal{F}^f$ which computes the intended function f and distributes the results to everyone.

Formally, we define a real-world and an ideal-world execution formally as below:

- $\mathbf{Real}^{\mathcal{A},\mathcal{Z},\Pi}(1^\kappa, 1^m, 1^s)$: $\mathcal{Z}(1^\kappa)$ chooses and provides inputs $x_1, x_2, \ldots, x_m$ for each of the m machines. Now the m machines engage in a protocol execution as explained above, where honest machines will faithfully execute the prescribed protocol Π using the input they obtained from $\mathcal{Z}$ but corrupt machines that are controlled by $\mathcal{A}$ can behave arbitrarily. At the end of the protocol, the honest machines send their output to $\mathcal{Z}$.
- $\mathbf{Ideal}^{\mathcal{S},\mathcal{Z},\mathcal{F}^f}(1^\kappa, 1^m, 1^s)$: The ideal-world execution involves the environment $\mathcal{Z}$ and an ideal-world adversary denoted $\mathcal{S}$. $\mathcal{Z}$ and $\mathcal{S}$ can communicate arbitrarily during the ideal-world execution. Now the execution is defined as below where $\mathsf{Honest} \subseteq [m]$ denotes the set of honest machines; and $\mathsf{Crupt} := [m] \backslash \mathsf{Honest}$ denotes the set of corrupt machines all of which are controlled $\mathcal{S}$:
  1. First, $\mathcal{Z}$ chooses and provides inputs $x_1, x_2, \ldots, x_m$ for each of the m machines.
  2. Every honest machine $i \in \mathsf{Honest}$ sends the input x_i received from $\mathcal{Z}$ to $\mathcal{F}^f$, and $\mathcal{F}^f$ records $\widetilde{x}_i := x_i$;
  3. For a corrupt machine $j \in \mathsf{Crupt}$, $\mathcal{F}^f$ may receive an input x'_j from j, and if so, it records $\widetilde{x}_j := x'_j$. Note that corrupt machines may use arbitrary inputs, and not necessarily the ones provided by $\mathcal{Z}$.
  4. As soon as all m machines have provided input, $\mathcal{F}^f$ computes the outputs $(y_1, y_2, \ldots, y_m) := f(\widetilde{x}_1, \widetilde{x}_2, \ldots, \widetilde{x}_m)$ and gives $\{y_i\}_{i \in \mathsf{Crupt}}$ to $\mathcal{S}$.
  5. Upon receiving `deliver` from the ideal-world adversary denoted $\mathcal{S}$, if any corrupt machine j has not yet provided input, set $\widetilde{x}_j := \perp$ and compute $(y_1, y_2, \ldots, y_m) := f(\widetilde{x}_1, \widetilde{x}_2, \ldots, \widetilde{x}_m)$. Now for $i \in [m]$, give y_i to machine i.
  6. Upon receiving an output from $\mathcal{F}^f$, an honest machine forwards the output to $\mathcal{Z}$.

In this paper, we define a notion of compositional security for multi-party computation analogous to the guarantees of Universal Composability [30]; moreover, our definition captures the requirement of *guaranteed output*, i.e., a small corrupted coalition should not be able to stop honest machines from producing output. The formal definition is provided below – note that the requirement of guaranteed output is captured since we require that the ideal-world adversary $\mathcal{S}$ send `deliver` to the ideal functionality:

▶ **Definition 21** (SMPC for MPC). *We say that an MPC protocol Π securely realizes some ideal functionality $\mathcal{F}^f$ against a t-bounded adversary where $t < m$ iff for any non-uniform polynomial-time adversary $\mathcal{A}$ that statically corrupts at most t out of m machines, there is a non-uniform polynomial-time ideal-world adversary $\mathcal{S}$ which is required to send* `deliver` *to $\mathcal{F}^f$, such that for any non-uniform polynomial-time environment $\mathcal{Z}$,*

$$\left\{ \mathsf{View}^{\mathcal{Z}}(\mathbf{Real}^{\mathcal{A},\mathcal{Z},\Pi}(1^\kappa, 1^m, 1^s)) \right\}_\kappa \approx \left\{ \mathsf{View}^{\mathcal{Z}}(\mathbf{Ideal}^{\mathcal{S},\mathcal{Z},\mathcal{F}^f}(1^\kappa, 1^m, 1^s)) \right\}_\kappa$$

where $\mathsf{View}^{\mathcal{Z}}(\mathbf{Expt})$ denotes the view of $\mathcal{Z}$ in the experiment $\mathbf{Expt}$ and $\approx$ denotes computational indistinguishability of two probability ensembles.

Resilience assumption

Henceforth we will assume that the adversary controls no more than $\frac{1}{3} - \eta$ fraction of the machines for an arbitrarily small constant η. Such a corruption threshold is (almost) the best one can hope for in a pairwise point-to-point network, since it takes at least 2/3 honest to realize broadcast [48, 82] (note that multi-party computation with guaranteed output implies broadcast).

5.3 Building Block: Constant-Round, Weakly Space-Efficient SMPC

We will leverage a universally composable SMPC protocol as a building block, and we would like this protocol to be not only constant round, but also somewhat efficient in space. Specifically, if the function $f(x_1, \ldots, x_{m'})$ evaluated requires $S \geq |x_1| + |x_2| + \ldots |x_{m'}|$ space to compute insecurely on a RAM, then for m' machines to securely compute the function f would require each machine to expend about $O(S)$ space, and moreover, allowing a $\mathsf{poly}(\kappa)$ blowup due to the use of cryptography. While this seems somewhat conserving in terms of space from the perspective of each individual machine, from the perspective of all m' machines, we are expending m' times more total space than the original RAM – for this reason, we call this notion weakly space-efficient. More formally, we require the following efficiency guarantees:

1. *Constant round*: the protocol must complete in $O(1)$ rounds;
2. *Weak space efficiency*: suppose that the function $f(x_1, x_2, \ldots, x_{m'})$ can be computed insecurely on a Random Access Machine (RAM) with S space where S must account for the space needed to write down all m' inputs and outputs, we would then like a protocol running on m' machines that securely realize $\mathcal{F}^f$, requiring only $\mathsf{poly}(\kappa) \cdot O(S)$ space on each machine.
3. *Communication efficiency*: total communication must be asymptotically not more than the total space of all machines;

Note that we cannot directly use this SMPC protocol to compile an MPC protocol to a secure counterpart if we want to preserve the efficiency of the original (insecure) MPC, since weak space efficiency still blows up each machine's space complexity to at least the size of the whole input. Looking ahead, we will run this SMPC building block within a randomly elected poly-logarithmic-sized committee to emulate a MPC machine with small space.

On the other hand, since a machine cannot receive messages of length greater than its space complexity, a constant-round, weakly space efficient SMPC protocol has communication complexity at most $\mathsf{poly}(\kappa) \cdot O(m' \cdot S)$, independent of the time complexity of the functionality f. Constant-round protocols based on garbled circuits or garbled RAM do not achieve desired efficiency. Thus, we consider FHE-based SMPC protocols, and furthermore, we would require the FHE schemes to satisfy a strong notion of *compactness* [55, 58] to avoid dependency on the circuit depth complexity of the functionality (which is the case if standard leveled FHE schemes are used). FHE schemes with this compactness property can be achieved by assuming circular security for standard FHE constructions [55,58] or using indistinguishability obfuscation [31, 53].

There is a long line of work on constant-round FHE-based SMPC that construct SMPC protocols based on threshold FHE (TFHE) or multi-key FHE (MKFHE) [14, 28, 72, 75, 91]. For our purpose, relying on these protocols would requires to assume above-mentioned compactness for TFHE or MKFHE (and trusted setup for some of them). Ideally, we would like to construct a protocol based on any compact (plain) FHE.

To achieve this goal, we follow the approach of constructing SMPC based on threshold FHE (e.g., [8]) and obtain the threshold FHE in use by applying the universal thresholdizer of Boneh et al. [23] to any compact FHE. We show that this yields a constant-round weakly space efficient SMPC protocol, but it requires a trusted setup. We further remove the setup by invoking another constant-round FHE-based SMPC protocol of Badrinarayanan et al. [14] to instantiate the setup. The protocol of [14] does not require a trusted setup. Furthermore, since the setup of threshold FHE has complexity independent of the complexity of the functionality, we do not need to assume compactness for the underlying (multi-key) FHE in the protocol of [14] to achieve weak space efficiency.

Later in our construction, we actually require that each machine receives different output in this SMPC protocol. Namely, the functionality to be securely computed is $f = (f_1, \ldots, f_{m'})$ where each machine M_i receives output $f_i(x_1, \ldots, x_{m'})$. In this case, we consider the space complexity as the maximal space complexity of $f_1, \ldots, f_{m'}$. Formally, we obtain the following theorem, which will serve as the building block of our SMPC for MPC construction. We prove the theorem in Appendix A.

▶ **Theorem 22** (Constant-round, weakly space-efficient, and communication efficient SMPC)**.** *Assume that the LWE assumption holds, the existence of enhanced trapdoor permutations, and the existence of FHE with an appropriate notion of compactness defined in Appendix A.1. Then, for any polynomial-time computable functions $f = (f_1, \ldots, f_{m'})$, there is a constant-round, weakly space-efficient, and communication efficient protocol that securely realizes $\mathcal{F}^f$ on m' machines against a t-bounded adversary as long as $m' \geq 3t + 1$.*

Proof. Deferred to Appendix A. ◀

5.4 Intuition

Given an original insecure MPC protocol that computes some function f over m machines' respective inputs, we would like to compile it to an SMPC protocol that securely realizes the functionality $\mathcal{F}^f$. We would like the compilation to be efficiency-preserving, that is, if the original MPC protocol completes in R rounds consuming s space per machine, then the compiled SMPC protocol completes in $O(R)$ rounds, and consumes $O(s) \cdot \mathsf{poly} \log \lambda \cdot \mathsf{poly}(\kappa)$ space per machine. Specifically, κ and λ denote a computational and a statistical security parameter respectively: the $\mathsf{poly}(\kappa)$ blowup is due to the use of cryptography and the $\mathsf{poly} \log \lambda$ blowup stems from random committee election.

Inspired by Boyle et al. [24, 27], our idea is to randomly elect a small, polylogarithmically sized committee to securely emulate each machine of the original MPC protocol. We use $m' = \mathsf{poly} \log \lambda$ to denote the size of each committee to distinguish from the total number of machines m. Suppose that $(1/3 - \eta)m$ machines are corrupt where η is an arbitrarily small constant, then by Chernoff bound, within each committee, only $t' \leq (1/3 - \eta/2)m'$ are corrupt except with negligible (in both λ and κ) probability.

Without loss of generality, henceforth we may assume that the original MPC protocol is communication-oblivious. If not, we can always take the compiler of Section 4.6 and compile the protocol to a communication-oblivious counterpart incurring only constant round and space blowup.

The state of each machine $i \in [m]$ in the original MPC protocol will now be secret shared among the i-th committee using a t'-out-of-m' robust secret sharing scheme. This means that at the beginning of the protocol, after each machine $i \in [m]$ receives an input it will invoke a protocol to secret share its input among the i-th committee. To accomplish this, machine i and the i-th committee will jointly perform a constant-round, weakly space-efficient,

and communication efficient SMPC (see Section 5.3) that realizes a robust secret-sharing functionality. Within each round, each machine $i \in [m]$ in the original MPC protocol must perform some local computation; in the compiled secure protocol, this computation will now be jointly performed by the i-th committee using a constant-round, weakly space-efficient, and communication efficient SMPC protocol (see Section 5.3). At end of this SMPC protocol, every committee member obtains a robust secret-share of machine i's new state in the original MPC. Finally, for a machine $i \in [m]$ to send a message to a machine $j \in [m]$ in the original MPC protocol, this communication will now also be implemented by an instance of a constant-round, weakly space-efficient SMPC protocol among the i-th and the j-th committees. At the end of the protocol, each member of the j-th committee should receive a robust secret share of the message.

5.5 Assumptions and Notations

5.5.1 Assumptions on the Original MPC

Without loss of generality, we can make the following assumptions on the original MPC to be compiled:

WLOG_1: We assume that the original MPC to be compiled has a deterministic communication pattern; and moreover, in every round every machine sends at most s words (where sending the same word to two machines is counted twice). Not only so, we may assume that in every round, every machine can compute on the fly and write down 1) an ordered list of at most s machines it wants to send words to and 2) an ordered list of at most s machines it is expecting to receive data from; and moreover this can be accomplished in $O(s)$ space.

If this is not the case, we can always apply the oblivious-compiler of Section 4.6 to make it so while incurring only constant blowup in round complexity and space.

WLOG_2: We may in fact assume that in the original MPC, at the end of the computation step in every round, every machine writes down at a designated location in memory (e.g., address 0) a list of at most s words to be sent. Recall that by WLOG_1, the destinations of these outgoing words are deterministic and a-priori known.

WLOG_3: We assume that in each round, after receiving messages from the network, a machine appends the received messages to its local memory in an arbitrary order. This is without loss of generality since during the computation step, the machine can always sort the received messages locally based on any order that is desired.

5.5.2 Notations

We will use the following notations:

- Let $\mathsf{mem}' \leftarrow M_r^i(\mathsf{mem})$ be the description of the RAM corresponding to machine i's computation in the r-th round in the original MPC; it takes in machine i's current memory mem and outputs a new memory state mem'.
- Let $m' = \mathsf{poly}\log\lambda$ denote each small committee's size, let η be an arbitrarily small constant, and let $t' = (1/3 - \eta/2)m'$ such that each committee has at most t' corrupt nodes except with negligible probability (see also the proof of Theorem 23).
- Let $(\mathsf{Share}, \mathsf{Recons})$ denote a t'-out-of-m' robust secret sharing scheme (see Appendix D). If the Share algorithm is provided with a string consisting of multiple words, we always assume that Share will perform secret sharing *word by word*; and similarly Recons will be performed word by word too.

- Later in our protocol, a small committee will be elected to emulate each machine in the original MPC protocol. Henceforth the committee that is emulating machine i in the original MPC is called the i-th committee.

5.5.3 Computing Relevant Committee Information on the Fly

To enable the pseudo-random committee election, a common reference string crs will be distributed to all honest machines at protocol start, and thus crs is common knowledge. In our protocol, committee election relies only on the crs. Specifically, the i-th committee is decided by $\mathsf{PRF}_{\mathsf{crs}}(i)$ where PRF is a pseudorandom function. At this point, it might seem safe to assume that the members of all committees are common knowledge. There is a slight subtlety here in that a machine in fact cannot store the members of all committees since this would consume too much space. Fortunately, by consuming $\mathsf{poly}\log\lambda \cdot O(s)$ additional space, a machine i can always compute on the fly and temporarily store members of committees relevant to itself in some round, including

1. which committees it is serving on, and all members of every committee it is serving on – we will show that every machine serves on at most $\Theta(m') = \mathsf{poly}\log\lambda$ committees except with negligible probability (see proof of Theorem 23);
2. all members of every committee it wants to communicate with in the present round: there are at most $2s$ such committees due to the s-sender-constraint and the s-receiver-constraint – note that to write this information down we rely on the fact that the original MPC to be compiled has a deterministic and fixed communication pattern;
3. all members of the committee emulating machine i itself.

Because of the above observations, later in Section 5.6, for simplicity it is unambiguous to parametrize our ideal functionalities that serve 1 or 2 committees with the relevant committee's indices – if a machine needs interact with some ideal functionality, it can be computed on-the-fly exactly which other machines will also be involved. Except with negligible probability, the additional per-machine space needed to compute on-the-fly and store the committee information relevant to the present round is bounded by $\mathsf{poly}\log\lambda \cdot O(s)$.

5.6 Intermediate Building Blocks

We will adopt the constant-round, weakly space efficient, and communication efficient SMPC protocol of Section 5.3 among one to two small committee(s) to securely realize a few useful ideal functionalities which we can adopt as intermediate building blocks:

- $\mathcal{F}^{\mathsf{share}[i]}$ is the ideal functionality that allows machine i to secret share its state among the i-th committee; $\mathcal{F}^{\mathsf{share}[i]}$ involves $m'+1$ participants among whom at most $t'+1$ can be corrupt;
- $\mathcal{F}^{\mathsf{comp}^r[i]}$ is the ideal functionality that allows the i-th committee to jointly emulate the computation of machine i in the r-th round in the original MPC; $\mathcal{F}^{\mathsf{comp}^r[i]}$ involves m' participants among whom at most t' corruptions; and
- $\mathcal{F}^{\mathsf{send}[i,i']}$ is the ideal functionality that emulates machine i sending a message to machine i' in the original MPC; $\mathcal{F}^{\mathsf{send}[i,i']}$ involves $2m'$ participants (i.e., the sending and receiving committees), among whom at most $2t'$ can be corrupt.

More formally, to define each of $\mathcal{F}^{\mathsf{share}[i]}$, $\mathcal{F}^{\mathsf{comp}^r[i]}$, and $\mathcal{F}^{\mathsf{send}[i,i']}$, we only need to specify what the functions $\mathsf{share}[i]$, $\mathsf{comp}^r[i]$, and $\mathsf{send}[i,i']$ compute respectively and among which players.

1. $\mathsf{share}[i]$: a function that anticipates inputs from machine i as well as members of the i-th committee. The function ignores everyone's input and looks at only machine i's input henceforth denoted x, and computes $(\overline{x}_1, \ldots, \overline{x}_{m'}) \leftarrow \mathsf{Share}(x)$. It outputs $\perp$ to machine i, and $\overline{x}_j$ to the j-th member of the i-th committee for $j \in [m']$.
2. $\mathsf{comp}^r[i]$: the function $\mathsf{comp}^r[i]$ anticipates inputs from members of the i-th committee denoted $\overline{\mathsf{mem}}_1, \overline{\mathsf{mem}}_2, \ldots, \overline{\mathsf{mem}}_{m'}$. It internally evaluates $\mathsf{mem}' \leftarrow M_r^i(\mathsf{Recons}(\{\overline{x}_j\}_{j\in[m']}))$, and $(\overline{\mathsf{mem}}'_1, \ldots, \overline{\mathsf{mem}}'_{m'}) \leftarrow \mathsf{Share}(\mathsf{mem}')$. Now, the j-th member of the i-th committee is supposed to get $\overline{\mathsf{mem}}'_j$ for $j \in [m']$.
3. $\mathsf{send}[i, i']$: this function anticipates inputs from the i-th committee and the i'-th committee. It ignores the inputs from the i'-th committee, and looks at only inputs from the i-th committee henceforth denoted $\overline{x}_1, \ldots, \overline{x}_{m'}$. It internally evaluates $x \leftarrow \mathsf{Recons}(\overline{x}_1, \ldots, \overline{x}_{m'})$, and $\overline{y}_1, \ldots, \overline{y}_{m'} \leftarrow \mathsf{Share}(x)$. Now, $\overline{y}_j$ is meant as the output for the j-th member of the i'-th committee for $j \in [m']$.

5.7 Compilation to a Hybrid Protocol

Since a machine may participate in multiple committees, henceforth when the machine we are concerned with is clear from the context, we often use the notation $\overline{\mathsf{mem}}_j$ a machine's robust secret share pertaining to the j-th committee (assuming that i indeed participates in the j-th committee). Given an original communication-oblivious MPC protocol, our SMPC compiler works as follows where $\mathsf{PRF} : \{0,1\}^\kappa \times [m] \to [m]^{m'}$ denotes a pseudo-random function:

- *Initialize.* At protocol start, a random common reference string denoted $\mathsf{crs} \xleftarrow{\$} \{0,1\}^\kappa$ is chosen which will be used for committee election. Every machine i now receives an input x_i from the environment $\mathcal{Z}$, and each machine is also informed of crs.
- *Committee election.* Now, the pseudo-random string $\mathsf{PRF}_{\mathsf{crs}}(j)$ can be used to determine the m' members of the j-th committee for $j \in [m]$. Note that instead of storing all members of every committee, relying on the observations in Section 5.5.3, machines will instead compute all the relevant committee information on the fly, in all of the subsequent steps of the protocol; and this will not consume too much space. For simplicity, in our description below, we will not explicitly describe how a machine computes the relevant committee information needed in every round.
- *Secret-share input.* Each machine $i \in [m]$ sends its input x_i to $\mathcal{F}^{\mathsf{share}[i]}$. For every committee i serves on, machine i sends $\perp$ to $\mathcal{F}^{\mathsf{share}[i]}$. As a result every member of the i-th committee obtains a robust secret share of x_i from $\mathcal{F}^{\mathsf{share}[i]}$. When a machine i participating in the j-th committee receives a robust secret share $\overline{v}$ from $\mathcal{F}^{\mathsf{share}[j]}$, it sets $\overline{\mathsf{mem}}_j := \overline{v}$.
- *Emulate protocol.* For every round $r \in [R]$ where R denotes the worst-case round complexity of the original MPC protocol, every machine $i \in [m]$ does the following:
  - *Emulate computation:* For each committee $j \in [m]$ the machine i serves on, machine i sends $\overline{\mathsf{mem}}_j$ to $\mathcal{F}^{\mathsf{comp}^r[i]}$. It will obtain from $\mathcal{F}^{\mathsf{comp}^r[i]}$ an updated share of the new memory denoted $\overline{\mathsf{mem}}'$. Machine i overwrites its $\overline{\mathsf{mem}}_j$ variable with $\overline{\mathsf{mem}}'$.
  - *Emulate sending:* For each committee $j \in [m]$ the machine i serves on, do the following: recall that by WLOG_2, shares of the words to be sent are written at a designated location in $\overline{\mathsf{mem}}_j$. By WLOG_1, the total number of words committee j wants to send in this round $s' \le s$ is deterministic and a-priori known; and moreover the s' destination machines can be computed on the fly and written down in $O(s)$ space. Henceforth let $d_1, d_2, \ldots, d_{s'}$ be the $s' \le s$ destination machines' identifiers, and let $\overline{y}_1, \ldots, \overline{y}_{s'}$ denote the shares of the words to send to them respectively. For each d_k where $k \in [s']$, send $\overline{y}_k$ to $\mathcal{F}^{\mathsf{send}[j,d_k]}$.

- *Emulate receiving:* For every committee j' that machine i serves on, if in the original MPC machine $j \in [m]$ is supposed to send message to machine j' (all such j's can be computed and written down on the fly due to WLOG_1), then for every such j:
  1. send $\perp$ to $\mathcal{F}^{\mathsf{send}[j,j']}$ indicating participation as a receiver;
  2. the machine i will receive a secret share $\overline{y}$ from $\mathcal{F}^{\mathsf{send}[j,j']}$, now append $\overline{y}$ to its $\overline{\mathsf{mem}}_j$.
- *Output.* After emulating all rounds of the original MPC in the above manner, machine i does the following: for every committee j it serves on, send $\overline{\mathsf{mem}}_j$ to machine j. When each machine i receives shares from at least $2m'/3$ members of the i-th committee, call Recons with the $2m'/3$ shares received and output the corresponding output.

▶ **Theorem 23.** *Suppose that the PRF scheme employed is secure*[8]*, and that the MPC protocol* Π *to be compiled obliviously realizes the ideal functionality* f *by Definition 6. Then, if we apply the above compiler to* Π *to obtain a hybrid-world protocol* Π_{hyb}*,* Π_{hyb} *must securely realizes* $\mathcal{F}^f$ *by Definition 21, as long as the adversary controls no more than* $1/3 - \eta$ *fraction of the machines (for an arbitrarily small constant* η*).*

Proof. Let $\mathcal{A}$ be a PPT adversary and $\mathsf{Crupt} \subset [m]$ of size $|\mathsf{Crupt}| \leq (1/3 - \eta) \cdot m$ be the set of machines corrupted by $\mathcal{A}$. We first show that except with negligible probability, (i) all the committees elected by $\mathsf{PRF}_{\mathsf{crs}}$ have at most $(1/3 - \eta/2) \cdot m'$ corrupted machines, and (ii) each machine participates in at most $2m'$ committees.[9] Suppose the committees were elected using truly uniform randomness, then the expected corrupted machines in a committee is $m' \cdot |\mathsf{Crupt}|/m \leq (1/3 - \eta) \cdot m'$ and the expected number of committees a machine participate is m'. By a Chernoff Bound and union bound, both (i) and (ii) hold except with probability $e^{-\Omega(m')} = \mathsf{negl}(\lambda)$. Since both properties can be checked efficiently, by the security of PRF, the probability that the committees elected by $\mathsf{PRF}_{\mathsf{crs}}$ violates violate (i) or (ii) is at most $\mathsf{negl}(\kappa) + \mathsf{negl}(\lambda)$. Hence, in the rest of the proof, we assume both (i) and (ii) hold. We proceed to define a simulator $\mathcal{S}$:

- *Initialize and Committee election.* $\mathcal{S}$ simply simulate a random crs. There is no communication in the committee election step.
- *Secret-share input.* In this step, $\mathcal{S}$ extracts the adversary $\mathcal{A}$'s input $\{x_i\}_{\mathsf{Crupt}}$ from $\mathcal{F}^{\mathsf{share}[i]}$ for $i \in \mathsf{Crupt}$. $\mathcal{S}$ sends $\{x_i\}_{\mathsf{Crupt}}$ to the ideal functionality. For the output of $\mathcal{F}^{\mathsf{share}[\cdot]}$ that $\mathcal{A}$ receives, since each committee has at most $(1/3 - \eta/2) \cdot m'$ corrupted machines, $\mathcal{S}$ can simulate the shares outputted by each $\mathcal{F}^{\mathsf{share}[\cdot]}$ by generating a fresh $\mathsf{Share}(0)$.
- *Emulate protocol.* Note both $\mathcal{F}^{\mathsf{comp}^r[\cdot]}$ and $\mathcal{F}^{\mathsf{send}[\cdot,\cdot]}$ also output shares, by the same reason, $\mathcal{S}$ can simulate the shares outputted by each $\mathcal{F}^{\mathsf{comp}^r[\cdot]}$ and $\mathcal{F}^{\mathsf{send}[\cdot,\cdot]}$ to $\mathcal{A}$ by fresh secret sharings $\mathsf{Share}(0)$.
- *Output.* $\mathcal{S}$ sends `deliver` to the ideal functionality and receives the output $\{y_i\}_{i \in \mathsf{Crupt}}$. For each $i \in \mathsf{Crupt}$, $\mathcal{S}$ generates $\mathsf{Share}(y_i)$ and sends the shares that machine i should receive from the honest machines in the i-th committee.

By perfect privacy of the RSS, it is clear that the shares outputted by $\mathcal{F}^{\mathsf{share}[\cdot]}$, $\mathcal{F}^{\mathsf{comp}^r[\cdot]}$ and $\mathcal{F}^{\mathsf{send}[\cdot,\cdot]}$ are simulated perfectly.

---

[8] In fact, this hybrid-world theorem secures against even computationally unbounded adversaries despite the use of the PRF, since the PRF is used to defeat only the polynomial checkable function whether some committee has 1/3 or more corruption.

[9] In fact, property (ii) is not needed for proving security but we will use it to analyze efficiency of the protocol later.

Now, note that our protocol Π_{hyb} emulate the underlying MPC protocol by committees, where all the input and messages are stored by shares of the robust secret sharing scheme (RSS) in each committee. Since all committee has at most $(1/3 - \eta) \cdot m' < m'/3$ corrupted machines, by robustness of RSS, the adversary cannot change the value stored in the RSS. Hence the underlying MPC protocol is emulated correctly, and at the end, each machine $i \in \mathsf{Crupt}$ receives shares of $\mathsf{Share}(y_i)$ from the i-th committee. Hence, $\mathcal{S}$ also simulates the shares in the output step perfectly.

Therefore, Π_{hyb} securely realizes $\mathcal{F}^f$, in fact, with statistical security in this hybrid model. ◀

5.8 Compilation to a Real-World Protocol

In Section 5.7, we compiled a communication-oblivious MPC protocol Π to a secure counterpart Π_{hyb} assuming the existence of ideal functionalities $\mathcal{F}^{\mathsf{share}[i]}$, $\mathcal{F}^{\mathsf{comp}^r[i]}$, and $\mathcal{F}^{\mathsf{send}[i,i']}$. Eventually we would like to replace these ideal functionalities with real-world building blocks. This is easy:

- Let $\Pi^{\mathsf{share}[i]}$, $\Pi^{\mathsf{comp}^r[i]}$, and $\Pi^{\mathsf{send}[i,i']}$ be SMPC protocols that securely realize (by Definition 21) $\mathcal{F}^{\mathsf{share}[i]}$, $\mathcal{F}^{\mathsf{comp}^r[i]}$, and $\mathcal{F}^{\mathsf{send}[i,i']}$ respectively. Note that $\Pi^{\mathsf{share}[i]}$ is a protocol among machine i and the i-th committee, $\Pi^{\mathsf{comp}^r[i]}$ is a protocol among the i-th committee, and $\Pi^{\mathsf{send}[i,i']}$ is a protocol among the i-th committee and the i'-th committee.
- In the compiled hybrid-world protocol in Section 5.7, whenever a machine invokes some ideal functionality $\mathcal{F}^{\mathsf{share}[i]}$, $\mathcal{F}^{\mathsf{comp}^r[i]}$, or $\mathcal{F}^{\mathsf{send}[i,i']}$ with the input x, it now invokes the corresponding protocol, $\Pi^{\mathsf{share}[i]}$, $\Pi^{\mathsf{comp}^r[i]}$, or $\Pi^{\mathsf{send}[i,i']}$ respectively, also with the input x.
- In the compiled hybrid-world protocol in Section 5.7, whenever a machine is to receive output from the ideal functionality $\mathcal{F}^{\mathsf{share}[i]}$, $\mathcal{F}^{\mathsf{comp}^r[i]}$, or $\mathcal{F}^{\mathsf{send}[i,i']}$, it now instead receives the output from $\Pi^{\mathsf{share}[i]}$, $\Pi^{\mathsf{comp}^r[i]}$, or $\Pi^{\mathsf{send}[i,i']}$ respectively.

Note that due to the observations made in Section 5.5.3, at the beginning of every round, every machine can compute on the fly and temporarily store members of all committees relevant to itself in this round, including committees it serves on and committees it will interact with – and this will only incur $O(s) \cdot \mathsf{poly}\log\lambda$ additional space. Therefore, for every protocol $\Pi^{\mathsf{share}[i]}$, $\Pi^{\mathsf{comp}^r[i]}$, or $\Pi^{\mathsf{send}[i,i']}$ invoked, the machine already knows exactly who are the other machines involved in the protocol. Not only so, in fact, at the beginning of every round, a machine has written down a receiving schedule for this round, again consuming $O(s) \cdot \mathsf{poly}\log\lambda$ additional space. As mentioned, if a machine receives any message from unanticipated senders or excessive messages from anticipated senders, these messages get discarded immediately and will not be stored or processed.

Efficiency of the compiled protocol

Let Π_{real} denote the compiled real-world protocol by applying the compiler described above to an original MPC communication-oblivious protocol Π. Since all of $\Pi^{\mathsf{share}[i]}$, $\Pi^{\mathsf{comp}^r[i]}$, and $\Pi^{\mathsf{send}[i,i']}$ complete in constant number of rounds, clearly, the round complexity of Π_{real} is only a constant factor more than the original Π.

We now analyze the per-machine space complexity. We will use the fact that $\Pi^{\mathsf{share}[i]}$, $\Pi^{\mathsf{comp}^r[i]}$, and $\Pi^{\mathsf{send}[i,i']}$ are weakly space efficient. We first focus on the space expended by each machine *for every committee it serves on*. Recall that each robust secret share of a machine's state in the original MPC is only $O(s)$ in size. The dominant part is the space consumed during $\Pi^{\mathsf{share}[i]}$, $\Pi^{\mathsf{comp}^r[i]}$, and $\Pi^{\mathsf{send}[i,i']}$ protocols. Note that all of the

functions evaluated by $\Pi^{\mathsf{share}[i]}$, $\Pi^{\mathsf{comp}^r[i]}$, and $\Pi^{\mathsf{send}[i,i']}$ protocols have at most $O(s)$ RAM-space complexity. Now, observe that a RAM consuming space s can be converted to a layered circuit of width s. Recall also that at most $2m'$ machines participate in each of $\Pi^{\mathsf{share}[i]}$, $\Pi^{\mathsf{comp}^r[i]}$, and $\Pi^{\mathsf{send}[i,i']}$. Thus by weak space efficiency, the space consumed by each $\Pi^{\mathsf{share}[i]}$, $\Pi^{\mathsf{comp}^r[i]}$, or $\Pi^{\mathsf{send}[i,i']}$ instance is at most $O(m' \cdot s) \cdot \mathsf{poly}(\kappa)$. So far we have focused on the space per machine per committee it serves on. The total space consumption of any single machine is at most $2m' \cdot O(m' \cdot s) \cdot \mathsf{poly}(\kappa)$, where $2m'$ is an upper bound on the number of committees a machine can participate in by the proof of Theorem 23 (except for a negligible probability). Since $m' = \mathsf{poly} \log \lambda$, the total space per machine is upper bounded by $\mathsf{poly} \log \lambda \cdot \mathsf{poly}(\kappa) \cdot O(s)$ (for some other suitable polynomial poly).

Finally, the total communication is asymptotically no more than the total space by the communication efficiency requirement.

▶ **Corollary 24.** *Suppose that the PRF employed is secure, and that the* $\Pi^{\mathsf{share}[i]}$, $\Pi^{\mathsf{comp}^r[i]}$, *and* $\Pi^{\mathsf{send}[i,i']}$ *protocols employed securely realize* $\mathcal{F}^{\mathsf{share}[i]}$, $\mathcal{F}^{\mathsf{comp}^r[i]}$, *and* $\mathcal{F}^{\mathsf{send}[i,i']}$ *protocols respectively by Definition 21 as long as at least* $2/3$ *fraction of the machines participating in each protocol instance are honest; and moreover suppose that they are constant-round, weakly space efficiency, and communication efficient as defined in Section 5.3. Suppose that the MPC protocol* Π *obliviously realizes the ideal functionality* f *by Definition 6. Then, if we apply the above compiler to* Π *to obtain a real-world protocol* Π_{real}, Π_{real} *must securely realizes* $\mathcal{F}^f$ *by Definition 21. Moreover,* Π_{real}*'s round complexity is asymptotically the same as* Π*; its per-machine space consumption is at most* $O(s) \cdot \mathsf{poly}(\kappa)$ *where* κ *denotes the security parameter and* s *is the per-machine space of the original* Π*, and its total communication is upper bounded* $O(m \cdot s) \cdot \mathsf{poly}(\kappa)$.

Proof. As shown in the proof of Theorem 23, indeed, in each committee at least $2/3$ fraction of the machines must be honest. Now, security follows from a standard compositional argument: for any real-world adversary $\mathcal{A}$ attacking Π_{real}, we can construct a hybrid-world adversary $\mathcal{A}'$ that basically calls the simulators of all instances of $\Pi^{\mathsf{share}[i]}$, $\Pi^{\mathsf{comp}^r[i]}$, and $\Pi^{\mathsf{send}[i,i']}$, and no polynomial-time environment $\mathcal{Z}$ should be able distinguish whether it is in the real world interacting with $\mathcal{A}$ or the hybrid world interacting with $\mathcal{A}'$. Now, by Theorem 23, for this hybrid-world adversary $\mathcal{A}'$ corresponding to the real-world adversary $\mathcal{A}$, we can construct an ideal-world adversary $\mathcal{S}$, such that no polynomial-time environment $\mathcal{Z}$ can distinguish whether it is in the hybrid world or the ideal world. Thus we can conclude that no polynomial-time environment $\mathcal{Z}$ can distinguish whether it is in the real world interacting with $\mathcal{A}$ or the ideal world interacting with $\mathcal{S}$.

Finally, the efficiency statements follow from the analysis in the paragraph before the corollary. ◀

Main theorem statement for SMPC-for-MPC

Recall that the main theorem statement for our "MPC to SMPC-for-MPC" compiler is Theorem 2. We restate it below for the reader's convenience and complete our presentation with a proof.

▶ **Theorem 25** (Secure computation for MPC: Restatement of Theorem 2)**.** *Assume the existence of a common random string, the Learning With Errors (LWE) assumption, enhanced trapdoor permutations, as well as the existence of an FHE scheme with a suitable notion of compactness (see Appendix A.1 for a formal definition of compactness). Suppose that* $s = N^{\varepsilon}$ *and that* m *is upper bounded by a fixed polynomial in* N*. Let* κ *denote a security parameter,*

and assume that $s \geq \kappa$. Given any MPC algorithm Π that completes in R rounds where each of the m machines has s local space, there is an MPC algorithm $\widetilde{\Pi}$ that securely realizes the same function computed by Π in the presence of an adversary that statically corrupts at most $\frac{1}{3} - \eta$ fraction of the machines for an arbitrarily small constant η. Moreover, $\widetilde{\Pi}$ completes in $O(R)$ rounds, consumes at most $O(s) \cdot \mathsf{poly}(\kappa)$ space per-machine, and incurs $O(m \cdot s) \cdot \mathsf{poly}(\kappa)$ total communication per round.

Proof. We may first apply the communication-oblivious compiler corresponding to Theorem 1 to compile Π to a communication-oblivious counterpart Π', we then apply the compiler corresponding to Corollary 24 on Π'. The theorem then follows in a straightforward fashion due to Theorem 1 and Corollary 24. ◀

References

1 Arash Afshar, Zhangxiang Hu, Payman Mohassel, and Mike Rosulek. How to Efficiently Evaluate RAM Programs with Malicious Security. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 702–729, 2015.

2 Kook Jin Ahn and Sudipto Guha. 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.

3 Prabhanjan Ananth, Yu-Chi Chen, Kai-Min Chung, Huijia Lin, and Wei-Kai Lin. Delegating RAM Computations with Adaptive Soundness and Privacy. In *Proceedings, Part II, of the 14th International Conference on Theory of Cryptography - Volume 9986*, 2016.

4 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 574–583, 2014. doi:10.1145/2591796.2591805.

5 Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel Graph Connectivity in Log Diameter Rounds. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 674–685, 2018.

6 Alexandr Andoni, Clifford Stein, and Peilin Zhong. Log Diameter Rounds Algorithms for 2-Vertex and 2-Edge Connectivity. *arXiv preprint*, 2019. arXiv:1905.00850.

7 Gilad Asharov, T-H. Hubert Chan, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Bucket Oblivious Sort: A Simple Oblivious Sort. In *SOSA*, 2019.

8 Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 483–501, 2012.

9 Gilad Asharov and Yehuda Lindell. A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation. *J. Cryptol.*, 30(1):58–151, January 2017.

10 Sepehr Assadi. Simple Round Compression for Parallel Vertex Cover. *CoRR*, abs/1709.04599, 2017.

11 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. *arXiv preprint*, 2017. arXiv:1711.03076.

12 Sepehr Assadi and Sanjeev Khanna. Randomized Composable Coresets for Matching and Vertex Cover. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 3–12. ACM, 2017.

13 Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. Massively Parallel Algorithms for Finding Well-Connected Components in Sparse Graphs. *CoRR*, abs/1805.02974, 2018. arXiv:1805.02974.

14 Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Threshold Multi-Key FHE and Applications to Round-Optimal MPC. Cryptology ePrint Archive, Report 2018/580, 2018.

15 Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5):454–465, 2012.

16 Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7):622–633, 2012.

17 MohammadHossein Bateni, Aditya Bhaskara, Silvio Lattanzi, and Vahab Mirrokni. Distributed balanced clustering via mapping coresets. In *Advances in Neural Information Processing Systems*, pages 2591–2599, 2014.

18 D. Beaver, S. Micali, and P. Rogaway. The Round Complexity of Secure Protocols. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 503–513, New York, NY, USA, 1990. ACM. doi:10.1145/100216.100287.

19 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, and Richard M. Karp. Massively Parallel Symmetry Breaking on Sparse Graphs: MIS and Maximal Matching. *CoRR*, abs/1807.06701, 2018.

20 Soheil Behnezhad, MohammadTaghi Hajiaghayi, and David G Harris. Exponentially Faster Massively Parallel Maximal Matching. *arXiv preprint*, 2019. arXiv:1901.03744.

21 Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

22 Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-cryptographic Fault-tolerant Distributed Computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, 1988.

23 Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold Cryptosystems from Threshold Fully Homomorphic Encryption. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, pages 565–596, 2018.

24 Elette Boyle, Kai-Min Chung, and Rafael Pass. Large-Scale Secure Computation: Multi-party Computation for (Parallel) RAM Programs. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 742–762, 2015.

25 Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious Parallel RAM and Applications. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, pages 175–204, 2016.

26 Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the Circuit Size Barrier for Secure Computation Under DDH. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 509–539, 2016.

27 Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication Locality in Secure Multi-party Computation: How to Run Sublinear Algorithms in a Distributed Setting. In *Proceedings of the 10th Theory of Cryptography Conference on Theory of Cryptography*, TCC'13, pages 356–376, Berlin, Heidelberg, 2013. Springer-Verlag. doi:10.1007/978-3-642-36594-2_21.

28 Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four Round Secure Computation without Setup. In *TCC*, 2017.

29 Sebastian Brandt, Manuela Fischer, and Jara Uitto. Matching and MIS for Uniformly Sparse Graphs in the Low-Memory MPC Model. *CoRR*, abs/1807.05374, 2018.

30 R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS*, 2001.

31 Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of Probabilistic Circuits and Applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography*, pages 468–497, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

32 Hubert Chan, Kai-Min Chung, and Elaine Shi. On the Depth of Oblivious Parallel ORAM. In *Asiacrypt*, 2017.

33 T.-H. Hubert Chan, Yue Guo, Wei-Kai Lin, and Elaine Shi. Cache-Oblivious and Data-Oblivious Sorting and Applications. In *SODA*, pages 2201–2220. SIAM, 2018.

34 T.-H. Hubert Chan, Kartik Nayak, and Elaine Shi. Perfectly Secure Oblivious Parallel RAM. In *Theory of Cryptography - 16th International Conference, TCC 2018*, pages 636–668, 2018.

35 T.-H. Hubert Chan and Elaine Shi. Circuit OPRAM: Unifying Statistically and Computationally Secure ORAMs and OPRAMs. In *Theory of Cryptography - 15th International Conference, TCC*, pages 72–107, 2017.

36 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The Complexity of (Delta+1) Coloring inCongested Clique, Massively Parallel Computation, and Centralized Local Computation. *arXiv preprint*, 2018. arXiv:1808.08419.

37 Yu-Chi Chen, Sherman S. M. Chow, Kai-Min Chung, Russell W. F. Lai, Wei-Kai Lin, and Hong-Sheng Zhou. Cryptography for Parallel RAM from Indistinguishability Obfuscation. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 179–190, 2016.

38 Kai-Min Chung and Luowen Qian. Adaptively Secure Garbling Schemes for Parallel Computations. In *TCC*, 2019.

39 Geoffroy Couteau. A Note on the Communication Complexity of Multiparty Computation in the Correlated Randomness Model. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pages 473–503, 2019.

40 Artur Czumaj, Jakub Łącki, Aleksander Mądry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. Round compression for parallel matching algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 471–484, 2018. doi:10.1145/3188745.3188764.

41 Rafael da Ponte Barbosa, Alina Ene, Huy L Nguyen, and Justin Ward. A New Framework for Distributed Submodular Maximization. In *FOCS*, pages 645–654, 2016.

42 Ivan Damgård and Yuval Ishai. Constant-round Multiparty Computation Using a Black-box Pseudorandom Generator. In *Proceedings of the 25th Annual International Conference on Advances in Cryptology*, CRYPTO'05, pages 378–394, Berlin, Heidelberg, 2005. Springer-Verlag.

43 Ivan Damgård and Yuval Ishai. Scalable Secure Multiparty Computation. In *Proceedings of the 26th Annual International Conference on Advances in Cryptology*, CRYPTO'06, pages 501–520, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11818175_30.

44 Ivan Damgård, Kasper Green Larsen, and Jesper Buus Nielsen. Communication Lower Bounds for Statistically Secure MPC, With or Without Preprocessing. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, pages 61–84, 2019.

45 Ivan Damgård, Jesper Buus Nielsen, Antigoni Polychroniadou, and Michael A. Raskin. On the Communication Required for Unconditionally Secure Multiplication. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 459–488, 2016.

46 Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using MapReduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 681–689. ACM, 2011.

47 Alina Ene and Huy Nguyen. Random coordinate descent methods for minimizing decomposable submodular functions. In *International Conference on Machine Learning*, pages 787–795, 2015.

48 Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy Impossibility Proofs for Distributed Consensus Problems. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, PODC '85, pages 59–70, New York, NY, USA, 1985. ACM. doi:10.1145/323596.323602.

49 Christopher Fletcher, Muhammad Naveed, Ling Ren, Elaine Shi, and Emil Stefanov. Bucket ORAM: Single Online Roundtrip, Constant Bandwidth Oblivious RAM. Cryptology ePrint Archive, Report 2015/1065, 2015. URL: `https://eprint.iacr.org/2015/1065`.

50 Christopher W. Fletcher, Ling Ren, Albert Kwon, Marten van Dijk, Emil Stefanov, and Srinivas Devadas. RAW Path ORAM: A low-latency, low-area hardware ORAM controller with integrity verification. *IACR Cryptology ePrint Archive*, 2014:431, 2014.

51 Christopher W. Fletcher, Ling Ren, Xiangyao Yu, Marten van Dijk, Omer Khan, and Srinivas Devadas. Suppressing the Oblivious RAM timing channel while making information leakage and program efficiency trade-offs. In *HPCA*, pages 213–224, 2014.

52 Buddhima Gamlath, Sagar Kale, Slobodan Mitrović, and Ola Svensson. Weighted Matchings via Unweighted Augmentations. *arXiv preprint*, 2018. `arXiv:1811.02760`.

53 Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate Indistinguishability Obfuscation and Functional Encryption for all circuits. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2013.

54 Sanjam Garg, Rafail Ostrovsky, and Akshayaram Srinivasan. Adaptive Garbled RAM from Laconic Oblivious Transfer. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, pages 515–544, 2018.

55 Craig Gentry. Fully homomorphic encryption using ideal lattices. In *ACM symposium on Theory of computing (STOC)*, 2009.

56 Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM Revisited. In *EUROCRYPT*, pages 405–422, 2014.

57 Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing Private RAM Computation. In *FOCS*, 2014.

58 Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

59 Mohsen Ghaffari. Massively Parallel Algorithms. `http://people.csail.mit.edu/ghaffari/MPA19/Notes/MPA.pdf`.

60 Mohsen Ghaffari, Themis Gouleakis, Slobodan Mitrovic, and Ronitt Rubinfeld. Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover. *CoRR*, abs/1802.08237, 2018.

61 Mohsen Ghaffari, Silvio Lattanzi, and Slobodan Mitrović. Improved Parallel Algorithms for Density-Based Network Clustering. In *International Conference on Machine Learning*, pages 2201–2210, 2019.

62 Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster Algorithms for Edge Connectivity via Random 2-Out Contractions, 2019. `arXiv:1909.00844`.

63 Mohsen Ghaffari and Jara Uitto. 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 2019, San Diego, California, USA, January 6-9, 2019*, pages 1636–1653, 2019.

64 O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *ACM Symposium on Theory of Computing (STOC)*, 1987.

65 O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *ACM symposium on Theory of computing (STOC)*, 1987.

66 Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 1996.

67 M. Goodrich. Communication-Efficient Parallel Sorting. *SIAM Journal on Computing*, 29(2):416–432, 1999.

68 Michael T. Goodrich. Data-oblivious External-memory Algorithms for the Compaction, Selection, and Sorting of Outsourced Data. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 379–388, New York, NY, USA, 2011. ACM. `doi:10.1145/1989493.1989555`.

69 Michael T. Goodrich and Michael Mitzenmacher. Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 576–587, 2011.

70 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, Searching, and Simulation in the MapReduce Framework. In *Algorithms and Computation*, pages 374–383, 2011.

71 S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

72 S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-Round MPC with Fairness and Guarantee of Output Delivery. In *CRYPTO*, pages 63–82, 2015.

73 Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-Efficient Unconditional MPC with Guaranteed Output Delivery. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, pages 85–114, 2019.

74 Jens Groth and Rafail Ostrovsky. Cryptography in the Multi-string Model. In *CRYPTO*, 2007.

75 Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a Chance of Partition Tolerance. In *CRYPTO*, 2019.

76 Sungjin Im, Benjamin Moseley, and Xiaorui Sun. Efficient massively parallel methods for dynamic programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 798–811, 2017. doi:10.1145/3055399.3055460.

77 Sungjin Im, Benjamin Moseley, and Xiaorui Sun. Efficient Massively Parallel Methods for Dynamic Programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 798–811, New York, NY, USA, 2017. ACM.

78 Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding Cryptography on Oblivious Transfer — Efficiently. In *CRYPTO*, 2008.

79 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948, 2010. doi:10.1137/1.9781611973075.76.

80 Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast Greedy Algorithms in MapReduce and Streaming. *TOPC*, 2(3):14:1–14:22, 2015. doi:10.1145/2809814.

81 Jakub Łącki, Vahab S. Mirrokni, and Michal Wlodarczyk. Connected Components at Scale via Local Contractions. *CoRR*, abs/1807.10727, 2018.

82 Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

83 Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*, pages 85–94, 2011. doi:10.1145/1989493.1989505.

84 Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient Constant-Round Multi-party Computation Combining BMR and SPDZ. *J. Cryptol.*, 32(3):1026–1069, July 2019.

85 Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. ObliVM: A programming framework for secure computation. In *IEEE Symposium on Security and Privacy*, 2015.

86 Steve Lu and Rafail Ostrovsky. How to Garble RAM Programs. In *EUROCRYPT*, 2013.

87 Steve Lu and Rafail Ostrovsky. Black-Box Parallel Garbled RAM. In *Advances in Cryptology – CRYPTO 2017*, pages 66–92, 2017.

88 Alfonso Cevallos Manzano. Reducing the Share Size in Robust Secret Sharing. Master's thesis, `http://algant.eu/documents/theses/cevallos.pdf`, 2011.

89 Vahab S. Mirrokni and Morteza Zadimoghaddam. Randomized Composable Core-sets for Distributed Submodular Maximization. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 153–162, 2015. `doi:10.1145/2746539.2746624`.

90 Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems*, pages 2049–2057, 2013.

91 Pratyay Mukherjee and Daniel Wichs. Two Round Multiparty Computation via Multi-key FHE. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 735–763, 2016.

92 Kartik Nayak, Xiao Shaun Wang, Stratis Ioannidis, Udi Weinsberg, Nina Taft, and Elaine Shi. GraphSC: Parallel Secure Computation Made Easy. In *IEEE S & P*, 2015.

93 Krzysztof Onak. Round Compression for Parallel Graph Algorithms in Strongly Sublinear Space. *CoRR*, abs/1807.08745, 2018.

94 Merav Parter and Eylon Yogev. Distributed Algorithms Made Secure: A Graph Theoretic Approach. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1693–1710, 2019.

95 Merav Parter and Eylon Yogev. Secure Distributed Computing Made (Nearly) Optimal. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, pages 107–116, New York, NY, USA, 2019. ACM. `doi:10.1145/3293611.3331620`.

96 Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. CacheShuffle: A Family of Oblivious Shuffles. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, 2018.

97 Vibhor Rastogi, Ashwin Machanavajjhala, Laukik Chitnis, and Anish Das Sarma. Finding connected components in map-reduce in logarithmic rounds. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 50–61, 2013.

98 Ling Ren, Xiangyao Yu, Christopher W. Fletcher, Marten van Dijk, and Srinivas Devadas. Design space exploration and optimization of path oblivious RAM in secure processors. In *ISCA*, pages 571–582, 2013.

99 Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and Circuits: (On Lower Bounds for Modern Parallel Computation). In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 1–12, 2016. `doi:10.1145/2935764.2935799`.

100 Elaine Shi, T.-H. Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM with $O((\log N)^3)$ Worst-Case Cost. In *ASIACRYPT*, pages 197–214, 2011.

101 Emil Stefanov, Elaine Shi, and Dawn Song. Towards Practical Oblivious RAM. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

102 Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM – an Extremely Simple Oblivious RAM Protocol. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

103 Leslie G. Valiant. A Bridging Model for Parallel Computation. *Commun. ACM*, 33(8):103–111, August 1990.

104 Xiao Shaun Wang, T-H. Hubert Chan, and Elaine Shi. Circuit ORAM: On Tightness of the Goldreich-Ostrovsky Lower Bound. In *CCS*, 2015.

105 Andrew Chi-Chih Yao. Protocols for Secure Computations (Extended Abstract). In *IEEE symposium on Foundations of Computer Science (FOCS)*, 1982.

106 Andrew Chi-Chih Yao. How to generate and exchange secrets. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1986.
107 Grigory Yaroslavtsev and Adithya Vadapalli. Massively Parallel Algorithms and Hardness for Single-Linkage Clustering under ℓ_p-Distances. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

A Proof of Theorem 22: the CommitteeMPC Protocol

In this section, we construct a constant-round, weakly space efficient, and communication efficient SMPC protocol as stated in Theorem 22. Starting from a compact FHE, we first apply the universal thresholdizer of Boneh et al. [23] to obtain a compact threshold FHE. We show that the resulting threshold FHE has desired security by Definition 21. Thus we can use it to construct a semi-malicious secure constant-round, weakly space efficient SMPC in a trusted setup model. The protocol can then be converted to a maliciously secure one by a generic transformation [14, 75], and the setup can be removed by invoking the SMPC protocol of Badrinarayanan et al. [14]. We start with the definitions.

Notation

We will use the variable m to denote the number of machines, although keep in mind that when the SMPC protocol in this section is employed in our "MPC to SMPC-for-MPC" compiler, this SMPC building block is in fact applied to at most $2m' = \mathsf{poly}\log\lambda$ number of machines.

A.1 Preliminaries

Fully Homomorphic Encryption

We first define fully homomorphic encryption schemes (FHE) with a strong compactness property. A FHE scheme is a tuple of PPT algorithms $\Pi_{\mathsf{FHE}} = (\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$ defined as follows:

- $\mathsf{FHE.KeyGen}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$: On input the security parameter κ, the key generation algorithm outputs a public key pk and a secret key sk.
- $\mathsf{FHE.Enc}(\mathsf{pk}, x) \to \mathsf{ct}$: On input a public key pk and a message $x \in \{0,1\}$, the encryption algorithm outputs a ciphertext ct. For convenience, for a message $x \in \{0,1\}^\ell$, we use $\mathsf{FHE.Enc}(\mathsf{pk}, x) = \mathsf{FHE.Enc}(\mathsf{pk}, x_1), \ldots \mathsf{FHE.Enc}(\mathsf{pk}, x_\ell)$ to denote the bit by bit encryptions of x.
- $\mathsf{FHE.Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \to \hat{\mathsf{ct}}$: On input a public key pk, a circuit $C : \{0,1\}^\ell \to \{0,1\}$ and ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$, the homomorphic evaluation algorithm outputs another ciphertext $\hat{\mathsf{ct}}$.
- $\mathsf{FHE.Dec}(\mathsf{sk}, \hat{\mathsf{ct}}) \to \hat{\mu}$: On input a secret key sk and a ciphertext $\hat{\mathsf{ct}}$, the decryption algorithm outputs a bit $\hat{\mu}$.

Correctness

We require that for all $\kappa \in \mathbb{N}$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa)$, circuit $C : \{0,1\}^\ell \to \{0,1\}$ and corresponding inputs $x_1, \ldots, x_\ell \in \{0,1\}$, it holds that

$$\Pr[\mathsf{FHE.Dec}(\mathsf{sk}, \mathsf{FHE.Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)) \neq C(x_1, \ldots, x_\ell)] \leq \mathsf{negl}(\kappa)$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.KeyGen}(1^k)$ and $\mathsf{ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, x_i)$.

Security

We require the usual semantic security. Namely, we require that for all $\kappa \in \mathbb{N}$, $(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, 0)) \approx_c (\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, 1))$, where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.KeyGen}(1^k)$.

Compactness

We require the following strong compactness property. There exists a polynomial poly such that the following holds. $|\mathsf{pk}|, |\mathsf{sk}|, |\mathsf{ct}| \leq \mathsf{poly}(\kappa)$ for the public and secret key, and any ciphertext ct generated from the algorithms of FHE. Furthermore, for a layered circuit[10] C with width w, homomorphic evaluation of C can be done in space $\mathsf{poly}(\kappa) \times w$, independent of the size or depth of the circuit.[11]

FHE schemes with this compactness property can be achieved by assuming circular security for standard FHE constructions [55, 58] or using indistinguishability obfuscation [31, 53].

Universal Thresholdizer

The following definition of universal thresholdizer is taken from Boneh et al. [23], who constructed universal thresholdizer based on the learning with error assumption. For our purpose, we do not require the verification algorithm, so we omit it from the definition for simplicity.

▶ **Definition 26.** *Fix a security parameter κ and a data space $\mathcal{X}$. A universal thresholdizer scheme is a tuple of algorithm $\Pi_{\mathsf{UT}} = (\mathsf{UT.Setup}, \mathsf{UT.Eval}, \mathsf{UT.Combine})$ defined as follows:*

- $\mathsf{UT.Setup}(1^\kappa, 1^m, 1^t, 1^d, x) \to (\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[m]})$*: On input the security parameter κ, a number of users in the system m, a threshold $t \in [m]$, a bound on the depth d, and a secret $x \in \mathcal{X}$, the setup algorithm generates the public parameters pp and a set of secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_m$ for each user in the system.*
- $\mathsf{UT.Eval}(\mathsf{pp}, \mathsf{sk}_i, C) \to p_i$*: On input the public parameters pp, a secret key sk_i, and a circuit C, the evaluation algorithm outputs a partial evaluation p_i.*
- $\mathsf{UT.Combine}(\mathsf{pp}, \{p_i\}_{i\in S}) \to \hat{\mu}$*: On input the public parameter pp, and a set of partial evaluations $\{p_i\}_{i\in S}$, the combining algorithm outputs the final evaluation μ.*

Evaluation Correctness

We say that a universal thresholdizer scheme $\Pi_{\mathsf{UT}} = (\mathsf{UT.Setup}, \mathsf{UT.Eval}, \mathsf{UT.Combine})$ satisfies evaluation correctness if the following conditions are true. For all $\kappa, m, t, d \in \mathbb{N}, x \in \mathcal{X}$, let $(\mathsf{pp}, \{sk_i\}_{i\in[m]}) \leftarrow \mathsf{UT.Setup}(1^\kappa, 1^m, 1^t, 1^d, x)$, $S \subset [m]$ of size $|S| = t$, and circuit $C : \mathcal{X} \to \{0, 1\}$ of depth at most d, we have that

$$\Pr[\mathsf{UT.Combine}(\mathsf{pp}, \{\mathsf{UT.Eval}(\mathsf{pp}, \mathsf{sk}_i, C)\}_{i\in S}) = C(x)] = 1 - \mathsf{negl}(\kappa),$$

where the probability is over the randomness of $\mathsf{UT.Setup}, \mathsf{UT.Eval}$, and $\mathsf{UT.Combine}$.

[10] A circuit is layered if the circuit can be represented as a layered graph with no wires crossing the layers.
[11] Here the space complexity measures the working space of $\mathsf{FHE.Eval}$ but not the description size of C. Looking ahead, we will consider uniform circuits whose description can be generated in small space.

Privacy

We say that a universal thresholdizer scheme $\Pi_{\mathsf{UT}} = (\mathsf{UT.Setup}, \mathsf{UT.Eval}, \mathsf{UT.Combine})$ satisfies privacy if there exists a PPT simulator Sim such that for all $\kappa \in \mathbb{N}$, polynomial m, t, d, PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, there exists a negligible function $\mathsf{negl}(\kappa)$ such that

$$\left|\Pr[\mathsf{Expt}^{\mathsf{Real}}_{\Pi_{\mathsf{UT}},\mathcal{A}}(\kappa, m, t, d) = 1] - \Pr[\mathsf{Expt}^{\mathsf{Rand}}_{\Pi_{\mathsf{UT}},\mathcal{A}}(\kappa, m, t, d) = 1]\right| \leq \mathsf{negl}(\kappa)$$

where the experiments $\mathsf{Expt}^{\mathsf{Real}}_{\Pi_{\mathsf{UT}},\mathcal{A}}$ and $\mathsf{Expt}^{\mathsf{Rand}}_{\Pi_{\mathsf{UT}},\mathcal{A}}$ are defined as follows:

- $\mathsf{Expt}^{\mathsf{Real}}_{\Pi_{\mathsf{UT}},\mathcal{A}}(\kappa, m, t, d)$:
  1. $(x^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\kappa)$.
  2. $(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[m]}) \leftarrow \mathsf{UT.Setup}(1^\kappa, 1^m, 1^t, 1^d, \underline{x^*})$.
  3. $(S^*, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{pp}, \mathsf{st}_1)$ where $|S^*| = t - 1$.
  4. $b \leftarrow \mathcal{A}_3^{\mathcal{O}_{\mathsf{Eval}}(\underline{\{\mathsf{sk}_i\}_{i\in[m]}}, \cdot, \cdot)}(\{\mathsf{sk}_i\}_{i\in S^*}, \mathsf{st}_2)$.
  5. Output b.
- $\mathsf{Expt}^{\mathsf{Rand}}_{\Pi_{\mathsf{UT}},\mathcal{A}}(\kappa, m, t, d)$:
  1. $(x^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\kappa)$.
  2. $(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[m]}) \leftarrow \mathsf{UT.Setup}(1^\kappa, 1^m, 1^t, 1^d, \underline{0^{|x^*|}})$.
  3. $(S^*, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{pp}, \mathsf{st}_1)$ where $|S^*| = t - 1$.
  4. $b \leftarrow \mathcal{A}_3^{\mathsf{Sim}^{\mathcal{O}_{\mathsf{Sim}}(\cdot)}(\underline{\{\mathsf{sk}_i\}_{i\in S^*}}, \cdot, \cdot)}(\{\mathsf{sk}_i\}_{i\in S^*}, \mathsf{st}_2)$.
  5. Output b.

where the oracles $\mathcal{O}_{\mathsf{Eval}}(\{\mathsf{sk}_i\}_{i\in[m]}, \cdot, \cdot)$ and $\mathcal{O}_{\mathsf{Sim}}(\cdot)$ are defined as follows

- $\mathcal{O}_{\mathsf{Eval}}(\{\mathsf{sk}_i\}_{i\in[m]}, C, j)$: On input the set of key $\{\mathsf{sk}_i\}_{i\in[m]}$, a circuit C, and an index $j \in [m]\backslash S^*$, outputs $\mathsf{UT.Eval}(\mathsf{pp}, \mathsf{sk}_j, C)$.
- $\mathcal{O}_{\mathsf{Sim}}(C)$: On input a circuit C, if there exists a query (C, j) for some $j \in [m]\backslash S^*$ previously made by $\mathcal{A}_3$, the algorithm outputs $C(x^*)$. Otherwise, it outputs $\perp$.

A.2 Threshold FHE

We now define a notion of threshold FHE schemes with a simulation security that is sufficient to directly construct a semi-malicious SMPC protocol in a trusted setup model. We then show that applying the above universal thresholdizer to a FHE scheme yields such a threshold FHE scheme.

Syntax

A threshold FHE scheme is a tuple of PPT algorithms $\Pi_{\mathsf{TFHE}} = (\mathsf{TFHE.Setup}, \mathsf{TFHE.SimSetup}, \mathsf{TFHE.Enc}, \mathsf{TFHE.Eval}, \mathsf{TFHE.PartDec}, \mathsf{TFHE.FinDec})$ defined as follows:

- $\mathsf{TFHE.Setup}(1^\kappa, 1^m, 1^t) \to (\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in[m]})$: On input the security parameter κ, a number of users in the system m, and a threshold $t \in [m]$, the setup algorithm generates the public key tpk and a set of secret keys $\mathsf{tsk}_1, \ldots, \mathsf{tsk}_m$ for each user in the system.
- $\mathsf{TFHE.SimSetup}(1^\kappa, 1^m, 1^t) \to (\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in[m]})$: On input the security parameter κ, a number of users in the system m, and a threshold $t \in [m]$, the simulation setup algorithm generates the simulated public key tpk and a set of simulated secret keys $\mathsf{tsk}_1, \ldots, \mathsf{tsk}_m$ for each user in the system.
- $\mathsf{TFHE.Enc}(\mathsf{tpk}, x) \to \mathsf{ct}$: On input a public key tpk and a message $x \in \{0, 1\}$, the encryption algorithm outputs a ciphertext ct.

- $\mathsf{TFHE.Eval}(\mathsf{tpk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \to \hat{\mathsf{ct}}$: On input a public key tpk, a circuit $C : \{0,1\}^\ell \to \{0,1\}$ and ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$, the homomorphic evaluation algorithm outputs another ciphertext $\hat{\mathsf{ct}}$.
- $\mathsf{TFHE.PartDec}(i, \mathsf{tpk}, \mathsf{tsk}_i, \hat{\mathsf{ct}}) \to p_i$: On input an index $i \in [m]$, a secret key tsk_i, and a ciphertext $\hat{\mathsf{ct}}$, the partial decryption algorithm outputs a partial decryption p_i.
- $\mathsf{TFHE.FinDec}(\mathsf{tpk}, \{p_i\}_{i\in S}) \to \hat{\mu}$: On input the public key tpk, and a set of partial decryptions $\{p_i\}_{i\in S}$, the final decryption algorithm outputs the final decryption value $\hat{\mu}$.

Correctness

We require that for all $\kappa, m, t \in \mathbb{N}$, $S \subset [m]$, circuit $C : \{0,1\}^\ell \to \{0,1\}$ and corresponding input $x_1, \ldots, x_\ell \in \{0,1\}$, the following holds except with negligible probability in κ: Let $(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in[m]}) \leftarrow \mathsf{TFHE.Setup}(1^\kappa, 1^m, 1^t)$, $\mathsf{ct}_i \leftarrow \mathsf{TFHE.Enc}(\mathsf{tpk}, x_i)$ for $i \in [\ell]$, let $\hat{\mathsf{ct}} = \mathsf{TFHE.Eval}(\mathsf{tpk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$, let $p_i \leftarrow \mathsf{TFHE.PartDec}(i, \mathsf{tpk}, \mathsf{tsk}_i, \hat{\mathsf{ct}})$, and $\hat{\mu} \leftarrow \mathsf{TFHE.FinDec}(\mathsf{tpk}, \{p_i\}_{i\in S})$. Then $\hat{\mu} = C(x_1, \ldots, x_\ell)$ if $|S| \geq t$, and $\hat{\mu} = \perp$ otherwise.

Compactness

We require the same compactness property as in FHE. There exists a polynomial poly such that the following holds. $|\mathsf{pk}|, |\mathsf{sk}|, |\mathsf{ct}| \leq \mathsf{poly}(\kappa)$ for the public and secret key, and any ciphertext ct generated from the algorithms of TFHE. Furthermore, for a layered circuit C with width w, homomorphic evaluation of C can be done in space $\mathsf{poly}(\kappa) \times w$, independent of the size or depth of the circuit.

Simulation Security

We require the following simulation-based security for the purpose of constructing SMPC protocols. There exists PPT algorithms $\mathsf{Sim}_1, \mathsf{Sim}_2$ such that for all κ, polynomial $m, t, s \in \mathbb{N}$ with $m \geq 3t + 1$, $S \subset [m]$ of size $|S| \leq t$, polynomial size circuits $C_j : \{0,1\}^{m\cdot s} \to \{0,1\}$ for $j \in S$, PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, there exists a negligible function $\mathsf{negl}(\kappa)$ such that

$$\left|\Pr[\mathsf{Expt}^{\mathsf{Real}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}(\kappa, m, t, S, \{C_j\}_{j\in S}) = 1] - \Pr[\mathsf{Expt}^{\mathsf{Ideal}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}(\kappa, m, t, S, \{C_j\}_{j\in S}) = 1]\right| \leq \mathsf{negl}(\kappa)$$

where the experiments $\mathsf{Expt}^{\mathsf{Real}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}$ and $\mathsf{Expt}^{\mathsf{Ideal}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}$ are defined as follows:

- $\mathsf{Expt}^{\mathsf{Real}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}(\kappa, m, t, S, \{C_j\}_{j\in S})$:
  1. $(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in[m]}) \leftarrow \mathsf{TFHE.Setup}(1^\kappa, 1^m, 1^t)$.
  2. $(\{x_i\}_{i\in[m]\setminus S}, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in S})$ where $x_i \in \{0,1\}^{\beta s}$.
  3. $\mathsf{ct}_i \leftarrow \mathsf{TFHE.Enc}(\mathsf{tpk}, x_i)$ for $i \in [m]\setminus S$.
  4. $(\{(x_i, r_i^{\mathsf{Enc}})\}_{i\in S}, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{st}_1, \{\mathsf{ct}_i\}_{i\in[m]\setminus S})$.
  5. $\mathsf{ct}_i \leftarrow \mathsf{TFHE.Enc}(\mathsf{tpk}, x_i; r_i^{\mathsf{Enc}})$ for $i \in S$; $\hat{\mathsf{ct}}_j \leftarrow \mathsf{TFHE.Eval}(\mathsf{tpk}, C_j, \{\mathsf{ct}_i\}_{i\in[m]})$ for $j \in S$.
  6. $p_{j,i} \leftarrow \mathsf{TFHE.PartDec}(i, \mathsf{tpk}, \mathsf{tsk}_i, \hat{\mathsf{ct}}_j)$ for $i \in [m]\setminus S$ and $j \in S$.
  7. $b \leftarrow \mathcal{A}_3(\mathsf{st}_2, \{p_{j,i}\}_{i\in[m]\setminus S, j\in S})$.
  8. Output b.
- $\mathsf{Expt}^{\mathsf{Ideal}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}(\kappa, m, t, S, \{C_j\}_{j\in S})$:
  1. $(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in[m]}) \leftarrow \mathsf{TFHE.SimSetup}(1^\kappa, 1^m, 1^t)$.
  2. $(\{x_i\}_{i\in[m]\setminus S}, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in S})$ where $x_i \in \{0,1\}^{\beta s}$.
  3. $(\{\mathsf{ct}_i\}_{i\in[m]\setminus S}, \mathsf{st}_1) \leftarrow \mathsf{Sim}_1(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in S})$
  4. $(\{(x_i, r_i^{\mathsf{Enc}})\}_{i\in S}, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{st}_1, \{\mathsf{ct}_i\}_{i\in[m]\setminus S})$.
  5. $\mathsf{ct}_i \leftarrow \mathsf{TFHE.Enc}(\mathsf{tpk}, x_i; r_i^{\mathsf{Enc}})$ for $i \in S$; $\hat{\mathsf{ct}}_j \leftarrow \mathsf{TFHE.Eval}(\mathsf{tpk}, C_j, \{\mathsf{ct}_i\}_{i\in[m]})$ for $j \in S$.

6. $\{p_{j,i}\}_{i\in[m]\setminus S, j\in S} \leftarrow \mathsf{Sim}_2(\mathsf{st}_1, \{(x_j, r_j^{\mathsf{Enc}})\}_{j\in S}, \{\hat{\mu}_j\}_{j\in S})$, where $\hat{\mu}_j = C_j(x_1, \ldots, x_m)$ for $j \in S$.
7. $b \leftarrow \mathcal{A}_3(\mathsf{st}_2, \{p_{j,i}\}_{i\in[m]\setminus S, j\in S})$.
8. Output b.

Construction

We show that applying the universal thresholdizer to a FHE scheme yields a threshold FHE scheme with above security. Let $\Pi_{\mathsf{FHE}} = (\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$ be a FHE scheme with circular security. Let $\Pi_{\mathsf{UT}} = (\mathsf{UT.Setup}, \mathsf{UT.Eval}, \mathsf{UT.Combine})$ be a universal thresholdizer. Formally, we construct a threshold FHE scheme as follows.

- $\mathsf{TFHE.Setup}(1^\kappa, 1^m, 1^t)$: Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa)$. Let d be the circuit depth of FHE decryption algorithm $\mathsf{FHE.Dec}$. Run $(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[m]}) \leftarrow \mathsf{UT.Setup}(1^\kappa, 1^m, 1^{t+1}, 1^d, \mathsf{sk})$. Let $\mathsf{tpk} = (\mathsf{pp}, \mathsf{pk})$ and $\mathsf{tsk}_i = \mathsf{sk}_i$ for $i \in [m]$. Output $(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in[m]})$.
- $\mathsf{TFHE.SimSetup}(1^\kappa, 1^m, 1^t)$: Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa)$. Let d be the circuit depth of FHE decryption algorithm $\mathsf{FHE.Dec}$. Run $(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[m]}) \leftarrow \mathsf{UT.Setup}(1^\kappa, 1^m, 1^{t+1}, 1^d, 0^{|\mathsf{sk}|})$. Let $\mathsf{tpk} = (\mathsf{pp}, \mathsf{pk})$ and $\mathsf{tsk}_i = \mathsf{sk}_i$ for $i \in [m]$. Output $(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in[m]})$.
- $\mathsf{TFHE.Enc}(\mathsf{tpk}, x)$: Output $\mathsf{ct} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, x)$.
- $\mathsf{TFHE.Eval}(\mathsf{tpk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$: Output $\hat{\mathsf{ct}} \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$.
- $\mathsf{TFHE.PartDec}(i, \mathsf{tpk}, \mathsf{tsk}_i, \hat{\mathsf{ct}})$: Output $p_i \leftarrow \mathsf{UT.Eval}(\mathsf{pp}, \mathsf{tsk}_i, \mathsf{FHE.Dec}(\cdot, \hat{\mathsf{ct}}))$.
- $\mathsf{TFHE.FinDec}(\mathsf{tpk}, \{p_i\}_{i\in S})$: Output $\hat{\mu} \leftarrow \mathsf{UT.Combine}(\mathsf{pp}, \{p_i\}_{i\in S})$.

It is clear by inspection that correctness follows by that of the underlying FHE scheme and universal thresholdizer. For compactness, note that universal thresholdizer is applied to evaluate the FHE decryption circuit $\mathsf{FHE.Dec}$, which has a fixed polynomial complexity in κ. Hence, the complexity of universal thresholdizer is upper bounded by a fixed $\mathsf{poly}(\kappa)$ and compactness follows by compactness of the underlying FHE scheme.

Security

We now show that the above construction satisfies simulation security defined above. We define simulators Sim_1, Sim_2 using the simulator of the universal thresholdizer (denoted by $\mathsf{UT.Sim}$) as follows.

- $\mathsf{Sim}_1(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in S})$: Simply run $\mathsf{ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, 0^{\beta s})$ for $i \in [m]\setminus S$, store $\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in S}$ in st_1, and output $(\{\mathsf{ct}_i\}_{i\in[m]\setminus S}, \mathsf{st}_1)$.
- $\mathsf{Sim}_2(\mathsf{st}_1, \{(x_i, r_i^{\mathsf{Enc}})\}_{i\in S}, \{\hat{\mu}_j\}_{j\in S})$: Run $p_{j,i} \leftarrow \mathsf{UT.Sim}^{\mathcal{O}_{\mathsf{Sim}}}(\{\mathsf{sk}_i\}_{i\in S}, C_j, i)$ for $i \in [m]\setminus S$ and $j \in S$, where $\mathcal{O}_{\mathsf{Sim}}$ on input query C' returns $\hat{\mu}_j$ if $C' = C_j$ for $j \in S$, and $\perp$ otherwise. Output $\{p_{j,i}\}_{i\in[m]\setminus S, j\in S}$.

Indistinguishability of $\mathsf{Expt}^{\mathsf{Real}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}$ and $\mathsf{Expt}^{\mathsf{Ideal}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}$ follows by considering a hybrid experiment that runs $\mathsf{TFHE.SimSetup}$ and Sim_2 in Step 1 and 6 respectively as the ideal experiment, but still encrypts x_i in Step 3. Formally, we define

- $\mathsf{Expt}^{\mathsf{Hyb}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}(\kappa, m, t, C, S)$:
  1. $(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in[m]}) \leftarrow \mathsf{TFHE.SimSetup}(1^\kappa, 1^m, 1^t)$.
  2. $(\{x_i\}_{i\in[m]\setminus S}, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i\in S})$ where $x_i \in \{0,1\}^{\beta s}$.
  3. $\mathsf{ct}_i \leftarrow \mathsf{TFHE.Enc}(\mathsf{tpk}, x_i)$ for $i \in [m]\setminus S$.
  4. $(\{(x_i, r_i^{\mathsf{Enc}})\}_{i\in S}, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{st}_1, \{\mathsf{ct}_i\}_{i\in[m]\setminus S})$.
  5. $\mathsf{ct}_i \leftarrow \mathsf{TFHE.Enc}(\mathsf{tpk}, x_i; r_i^{\mathsf{Enc}})$ for $i \in S$; $\hat{\mathsf{ct}}_j \leftarrow \mathsf{TFHE.Eval}(\mathsf{tpk}, C_j, \{\mathsf{ct}_i\}_{i\in[m]})$ for $j \in S$.
  6. $\{p_{j,i}\}_{i\in[m]\setminus S, j\in S} \leftarrow \mathsf{Sim}_2(\mathsf{st}_1, \{(x_j, r_j^{\mathsf{Enc}})\}_{j\in S}, \{\hat{\mu}_j\}_{j\in S})$, where $\hat{\mu}_j = C_j(x_1, \ldots, x_m)$ for $j \in S$.
  7. $b \leftarrow \mathcal{A}_3(\mathsf{st}_2, \{p_{j,i}\}_{i\in[m]\setminus S, j\in S})$.
  8. Output b.

We claim that indistinguishability of $\mathsf{Expt}^{\mathsf{Real}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}$ and $\mathsf{Expt}^{\mathsf{Hyb}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}$ follows directly by privacy of universal thresholdizer. Indeed, observe that the difference between $\mathsf{TFHE.Setup}$ and $\mathsf{TFHE.SimSetup}$ is in the call of $\mathsf{UT.Setup}$, where$\mathsf{TFHE.Setup}$ uses actual FHE secret key sk and $\mathsf{TFHE.SimSetup}$ uses $0^{|\mathsf{sk}|}$. Also in Step 6, partial decryptions $\mathsf{TFHE.PartDec}$ of $\hat{\mathsf{ct}}_j$ for $j \in S$, which in turn are the partial evaluations $\mathsf{UT.Eval}$ on the FHE decryption circuit $\mathsf{FHE.Dec}(\cdot, \mathsf{sk})$, is replaced by the simulator of the universal thresholdizer $\mathsf{UT.Sim}$, where $\mathcal{O}_{\mathsf{Sim}}$ is emulated correctly by returning $\hat{\mu}_j = C_j(x_1, \ldots, x_m)$ when queried by C_j for $j \in S$. Hence, $\mathsf{Expt}^{\mathsf{Real}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}$ and $\mathsf{Expt}^{\mathsf{Hyb}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}$ correspond to $\mathsf{Expt}^{\mathsf{Real}}_{\Pi_{\mathsf{UT}},\mathcal{A}}$ and $\mathsf{Expt}^{\mathsf{Rand}}_{\Pi_{\mathsf{UT}},\mathcal{A}}$ for universal thresholdizer, and the indistinguishability follows by privacy of universal thresholdizer.

Now, observe that the difference between $\mathsf{Expt}^{\mathsf{Hyb}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}$ and $\mathsf{Expt}^{\mathsf{Ideal}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}$ is only the messages encrypted in Step 3 and that the FHE secret key is not used in the experiments. Hence, indistinguishability of $\mathsf{Expt}^{\mathsf{Hyb}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}$ and $\mathsf{Expt}^{\mathsf{Ideal}}_{\Pi_{\mathsf{TFHE}},\mathcal{A}}$ follows directly by semantic security of FHE. This completes the proof of security for the constructed TFHE scheme.

A.3 Semi-Malicious Secure SMPC in a Trusted Setup Model

We proceed to construct a semi-malicious secure constant-round, weakly space efficient SMPC in a trusted setup model using threshold FHE. We consider SMPC protocols over m machines. Henceforth let β denote the bit-width of each word. Each machine holds input $x_i \in \{0,1\}^{\beta s}$ and wishes to learn $f_i(x_1, \ldots, x_m)$ for functions $f_i : \{0,1\}^{m \cdot \beta \cdot s} \to \{0,1\}$.[12] The construction is rather straightforward: We use the setup to run the threshold FHE setup algorithm $\mathsf{TFHE.Setup}$ and distribute the keys. Upon receiving the keys, each machine encrypts its input and output the ciphertext. Then they locally evaluate the output ciphertexts homomorphically, partially decrypt them, and send the partial decryptions to the corresponding machines, who can then learn their own outputs.

Formally, let $\Pi_{\mathsf{TFHE}} = (\mathsf{TFHE.Setup}, \mathsf{TFHE.SimSetup}, \mathsf{TFHE.Enc}, \mathsf{TFHE.Eval}, \mathsf{TFHE.PartDec}, \mathsf{TFHE.FinDec})$ be a threshold FHE scheme. We construct the following SMPC protocol Π_{SMPC} over m machines. Let t be an upper bound on the number of corrupted machines, where $m \geq 3t + 1$.

- **Input and functionality**:
  - Each machine M_i has input $x_i \in \{0,1\}^{\beta s}$ and wishes to learn $f_i(x_1, \ldots, x_m)$ for functions $f_i : \{0,1\}^{m \cdot \beta \cdot s} \to \{0,1\}$.
- **Setup Stage**:
  - Run $(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i \in [m]}) \leftarrow \mathsf{TFHE.Setup}(1^\kappa, 1^m, 1^t)$.
  - Send $(\mathsf{tpk}, \mathsf{tsk}_i)$ to each machine M_i for $i \in [m]$.
- **Round** 1: Each machine M_i does the following:
  - Run $\mathsf{ct}_i \leftarrow \mathsf{TFHE.Enc}(\mathsf{tpk}, x_i)$.
  - Broadcast ct_i.
- **Round** 2: Each machine M_i does the following:
  - Record each ct_j received from machine M_j. If M_j aborts, then use $\mathsf{ct}_j \leftarrow \mathsf{TFHE.Enc}(\mathsf{tpk}, 0^{\beta s}; 0^*)$ as a default ciphertext.
  - Compute $\hat{\mathsf{ct}}_j \leftarrow \mathsf{TFHE.Eval}(\mathsf{tpk}, C_j, \{\mathsf{ct}_i\}_{i \in [m]})$ for $j \in [m]$.
  - Compute $p_{j,i} \leftarrow \mathsf{TFHE.PartDec}(i, \mathsf{tpk}, \mathsf{tsk}_i, \hat{\mathsf{ct}}_j)$ for $j \in [m]$.
  - Send $p_{j,i}$ to machine M_j for each $j \in [m]$.

[12] For notational simplicity, we consider functions with one-bit output. It is straightforward to extend the protocol to handle long outputs.

- **Output Computation**: Each machine M_i does the following:
  - Record each $p_{i,j}$ received from machine M_j. Let S_i be the set of partial decryptions received by M_i.
  - Compute $\hat{\mu}_i \leftarrow \mathsf{TFHE.FinDec}(\mathsf{tpk}, \{p_{i,j}\}_{j \in S_i})$.
  - Output $\hat{\mu}_i$.

Correctness

Correctness follows directly from the correctness of TFHE. Furthermore, since the number of honest machines is greater than $t+1$, the honest machines always learn the output.

Weak Space Efficiency

We first note that by compactness, except for the homomorphic evaluation TFHE.Eval, the remaining step takes space at most $\mathsf{poly}(\kappa) \cdot O(ms)$. Let S_{RAM} denote the space complexity for computing $f_1, \ldots, f_m$ using a RAM machine. It is not hard to see that we can emulate the RAM computation by a uniform layered circuit of width $O(S_{\mathsf{RAM}})$: for example, a naïve way is to emulate CPU and memory by circuits and for each RAM computation step, emulate CPU accessing memory by constructing a linear-sized circuit gadget that goes over every memory cell to selects the position requested. Thus, it follows by compactness that TFHE.Eval can be done in space $\mathsf{poly}(\kappa) \cdot O(S_{\mathsf{RAM}})$.

Semi-malicious Security

The security follows directly from the simulation security of TFHE. For completeness, we formally define the simulator $\mathcal{S}$ for Π_{SMPC} as follows. Let Honest and Crupt denote the set of honest and corrupted machines, respectively.

- **Setup Stage**: $\mathcal{S}$ simulates the setup by running TFHE.SimSetup instead of TFHE.Setup. Namely,
  - Run $(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i \in [m]}) \leftarrow \mathsf{TFHE.SimSetup}(1^\kappa, 1^m, 1^t)$.
  - Send $\{(\mathsf{tpk}, \mathsf{tsk}_i)\}_{i \in \mathsf{Crupt}}$ to the adversary $\mathcal{A}$.
- **Round 1**:
  - $\mathcal{S}$ simulates the honest machines' messages by running Sim_1 of TFHE. Namely, $\mathcal{S}$ runs $(\{\mathsf{ct}_i\}_{i \in \mathsf{Honest}}, \mathsf{st}_1) \leftarrow \mathsf{Sim}_1(\mathsf{tpk}, \{\mathsf{tsk}_i\}_{i \in \mathsf{Crupt}})$. $\mathcal{S}$ sends $\{\mathsf{ct}_i\}_{i \in \mathsf{Honest}}$ to $\mathcal{A}$.
  - Upon receiving $(\{(x_i, r_i^{\mathsf{Enc}})\}_{i \in \mathsf{Crupt}}$ from $\mathcal{A}$, $\mathcal{S}$ sends it to the ideal functionality. If any corrupted machine i aborts, then $\mathcal{S}$ sends $x_i = 0^{\beta s}$ to the ideal functionality.
  - $\mathcal{S}$ receives the outputs $\{\hat{\mu}_i\}_{i \in \mathsf{Crupt}}$ from the ideal functionality.
- **Round 2**:
  - $\mathcal{S}$ simulates the honest machines' messages by running Sim_2 of TFHE. Namely, $\mathcal{S}$ runs $\{p_{j,i}\}_{i \in \mathsf{Honest}, j \in \mathsf{Crupt}} \leftarrow \mathsf{Sim}_2(\mathsf{st}_1, \{(x_j, r_j^{\mathsf{Enc}})\}_{j \in \mathsf{Crupt}}, \{\hat{\mu}_j\}_{j \in \mathsf{Crupt}})$. $\mathcal{S}$ sends $\{p_{j,i}\}_{i \in \mathsf{Honest}, j \in \mathsf{Crupt}}$ to $\mathcal{A}$.
  - $\mathcal{S}$ sends `deliver` to the ideal functionality.

It is not hard to see that the real world and ideal world execution of Π_{SMPC} directly corresponds to the experiments $\mathsf{Expt}^{\mathsf{Real}}_{\Pi_{\mathsf{TFHE}}, \mathcal{A}}$ and $\mathsf{Expt}^{\mathsf{Ideal}}_{\Pi_{\mathsf{TFHE}}, \mathcal{A}}$ in the simulation security of TFHE with corresponding adversary. Hence, indistinguishability of the simulation for Π_{SMPC} follows by the simulation security of TFHE.

A.4 Achieving Malicious Security and Removing the Trusted Setup

Finally, we briefly discuss how to upgrade to malicious security and remove the trusted setup in Π_{SMPC}. To upgrade to malicious security, we can apply the standard generic transformation using a simulation-extractable multi-string NIZK which can be constructed from enhanced trapdoor permutations without extra setup [14, 74, 75]. Note that NIZK is used to prove that $\mathsf{TFHE.Enc}$ and $\mathsf{TFHE.PartDec}$ are done correctly, both statements have a fixed $\mathsf{poly}(\kappa)$ complexity. The NIZK proofs can be generated in a fixed $\mathsf{poly}(\kappa)$ space as well. Thus, the transformation preserves weak space efficiency.

To remove the setup, we rely on an SMPC protocol by Badrinarayanan et al. [14]. The protocol of [14] (Theorem 10) is constant-round, achieves guaranteed output delivery, and does not require any setup. We remove the trusted setup by invoking the protocol of Badrinarayanan et al. [14] to securely realize the setup stage in Π_{SMPC}. Note that $\mathsf{TFHE.Setup}$ has a fixed $\mathsf{poly}(\kappa)$ complexity (independent of the functionalities $f_1, \ldots, f_m$). As a result, we obtain a malicious security, constant-round, weakly space efficient and communication efficient SMPC protocol, as required in Theorem 22.

B Potential Barriers Towards Achieving Statistical Security

We have shown how to compile an MPC protocol to a secure counterpart that defends against slightly less than 1/3 corruption while preserving its efficiency; but our compiler relied on a few computational assumptions such as enhanced trapdoor permutations, LWE, and an appropriate notion of compact FHE. One intriguing question is whether we can accomplish the same, but *unconditionally*, i.e., without making any cryptographic hardness assumptions. Such protocols are also said to be *statistically* secure. We now show that if one could achieve the same result unconditionally, it will imply solutions to long-standing open questions in cryptography. Specifically, we prove the following theorem:

▶ **Theorem 27.** *Let κ denote the security parameter. Suppose that there exists an compiler that compiles any MPC protocol Π computing the function f among m machines into an SMPC-for-MPC protocol Π' among m machines that securely realizes $\mathcal{F}^f$ unconditionally, as long as m is polynomially bounded by s and $s \geq \kappa$. Furthermore, suppose that the compiler incurs only $O(1)$ blowup in round complexity and $\mathsf{poly}(\kappa)$ blowup in terms of per-machine space complexity.*

Then, for $m \geq \kappa$, for any m-input, m-output uniform layered circuit C with width m, as long as C's size is a sufficiently large polynomial in m (related to the parameter α later), then there exists a constant-round protocol that allows m parties to securely realize $\mathcal{F}^C$ unconditionally, incurring total communication that is $|C|^\alpha$ for an arbitrarily small constant $\alpha \in (0, 1)$.

Proof. If such a compiler existed, we can use it to compile the following insecure 1-round MPC protocol among m machines each of which has $s = O(m)$ space. Every one now sends their input to the first machine, the first machine computes the circuit C locally, and sends to each machine their respective output. Note that since the circuit is uniform and layered with maximum width m, the first machine can evaluate it in total space $O(m)$.

Now consider the compiled protocol: it will complete in $O(1)$ rounds, and moreover the total communication must be upper bounded by the number of rounds multiplied by $m \cdot O(s) \cdot \mathsf{poly}(\kappa) = O(m^2)$, which can be made $|C|^\alpha$ as long as the circuit size is a sufficiently large polynomial in m where $|C|$ denotes the size of C. ◀

As noted in numerous works in the cryptography literature [26, 39, 44, 45], the existence of such constant-round, sublinear-communication multi-party computation (for circuits) with *statistical* security has been a long-standing open problem, even for the special class of circuits that we consider. To the best of our knowledge, the best known n-party, statistically secure computation protocol achieves the following[13]

- $O(n|C|)$ total communication and d number of rounds without preprocessing [43, 73] where d denotes the circuit depth, and
- $O(n|C|/\log\log|C|)$ total communication and $d/\log\log|C|$ number of rounds with (polynomially-bounded) preprocessing (for layered circuits) [39].

Interestingly, we note that barring strong assumptions such as Indistinguishable Obfuscation [53], the only known approach to construct constant-round, sublinear-communication *multi*-party computation for circuits of unbounded polynomial size is also through compact FHE [55, 58]. In our earlier sections, we essentially showed that making a similar assumption, combined with other standard cryptographic assumptions, we can construct an efficiency-preserving "MPC to SMPC-for-MPC" compiler. From a technical perspective, the main new challenge we encountered is the fact that the machines are now also *space-constrained* (which was not a concern in the standard multi-party computation for circuit literature), and thus we could not just apply existing techniques to the entire set of machines.

Besides the feasibility of achieving statistical security, another interesting direction is to weaken the cryptographic assumption necessary in achieving such a compiler. Similarly, new results in this vein would imply new breakthroughs for constant-round, sublinear-communication *multi*-party computation for circuits too – and even partial results for special classes of circuits (like the family we considered in Theorem 27) would be interesting.

C Removing the Sender Constraint

As mentioned earlier, some works in the MPC literature do not seem to respect the s-sender-constraint, and only respect the s-receiver-constraint. In this section, we generalize our results even to such MPC algorithms.

To achieve this, it suffices to show that given an MPC protocol denoted Π that respects only the s-receiver-constraint, we can compile it to a counterpart denoted Π' that satisfies not just the $O(s)$-receiver-constraint, but also the $O(s)$-sender-constraint. Furthermore, the compilation should preserve both round- and space-complexity; and moreover, the compiled protocol Π' should run in a *fixed* number of rounds (since the oblivious Routing primitive applied to emulate the communication of Π' will not hide total round complexity).

Intuition

The idea is the following: in the first phase (called the replication phase), if a sender wants to send in total μ words in some communication round (where sending the same word to two machines is counted twice), it will *replicate* all of its local memory to $\lceil\mu/s\rceil$ helper-machines where each helper is assigned a unique index from 1 to $\lceil\mu/s\rceil$. This must be accomplished using an s-sender-bounded communication pattern. In the second phase (called the distribution phase), each helper distributes s words on behalf of the sender it represents. Note that in the same round, many machines may be trying to send data simultaneously.

[13] For this reason, in fact even if we relaxed the round complexity blowup in Theorem 27 to poly-logarithmically many rounds or any number that does not depend on circuit depth, having such an MPC-to-SMPC compiler would still imply improving the state-of-the-art for statistically secure MPC.

Thus the above procedure is performed in parallel among all senders. It must be guaranteed that *every machine serves as a helper at most for one sender*. In this way, the distribution phase will satisfy the s-sender-constraint.

C.1 Replication Protocol

A replication protocol allows senders to replicate their local memory to an appropriate number of helpers.

Definition

Formally, replication, henceforth denoted Replicate, is the following problem.

- **Input**. Suppose that among the m machines, some machines are senders and others are non-senders. Each machine i obtains an input pair (β_i, c_i) where $\beta_i \in \{0, 1\}$ is a bit indicating whether machine i is a sender; and if $\beta_i = 1$, $c_i \geq 1$ denotes the total number of machines to replicate machine i's state – henceforth we refer to c_i as sender i's *multiplicity*.

  It is guaranteed that $\sum_{i=1}^{m} \beta_i c_i \leq m$, i.e., in total there are enough machines around to act as receivers.
- **Output**. At the end of the replication protocol, the following output configuration is produced:
  - each sender i has its entire machine state (i.e., a total of s words) replicated on exactly c_i receivers;
  - each machine acts as a receiver for at most 1 sender;
  - suppose sender i has c_i receivers, each of these receivers output i and also a unique index j from the range $[c_i]$, i.e., each of these receiver knows that it acts as the j-th receiver for its sender.

We will next construct a Replicate protocol. Note that the protocol need not be communication-oblivious. The idea is that all machines will first perform a prefix sum computation which allows each sender to discover a range of indices which are meant to become its helpers; moreover, all senders' helpers, identified by the range, are disjoint. It is well-known that prefix sum can be accomplished on MPC in $O(1)$ rounds with an s-sender-bounded communication style. Now, we employ a RangeCast protocol for the sender to replicate its state to the range of machines discovered above. Below we first explain how to construct the RangeCast building block and then describe our Replicate protocol.

Building block: RangeCast

As mentioned, RangeCast allows a sender to replicate its state to a set of machines defined by a range $[a, b] \subseteq [m]$. To realize such a RangeCast primitive, we first realize a weaker form denoted WeakRangeCast which only works if the range's size is at most s. Our RangeCast is similar to the "broadcast" algorithm in the "bulk-synchronous parallel (BSP)" model [103], but we describe it again for completeness.

WeakRangeCast

Input: let a, b be any two machines such that $1 \leq a \leq b \leq m$, let I be an array of s words. The machine a receives the input (I, a, b) where *the range* $[a, b]$ *is small enough such that* $b - a + 1 \leq s$.

Output: every machine $k \in [a, b]$ outputs I.

Protocol:

1. Let $c := b - a + 1$ and $t := \lceil s/c \rceil$. For each $j \in [c]$, let $I_j := I[(j-1)t + 1 : jt]$ be the substring of I where I_j consists of at most t words (if jt or $(j-1)t+1$ is less than s, I_j is by definition a shorter or empty string). In this round, the machine a sends to the machine $a + j - 1$ the message tuple (a, b, I_j) for each $j \in [c]$, while all other machines send nothing.
2. In the next round, machine $a + j - 1$ receives (a, b, I_j) for each $j \in [c]$, and it sends the same message (j, I_j) to every machine $k \in [a, b]$.
3. Every machine $k \in [a, b]$ receives a copy of (j, I_j) for all $j \in [c]$, and it recovers I by concatenating $(I_j)_{j \in [c]}$.

▶ **Lemma 28.** *For any* $1 \leq a \leq b \leq m$ *such that* $b - a + 1 \leq s$, *WeakRangeCast correctly implements range-cast in 2 rounds and satisfies both* $O(s)$*-sender- and* $O(s)$*-receiver-constraints, where each machine takes* $O(s)$ *space and time locally.*

Proof. The correctness, rounds, and local-machine complexity follows directly. The $O(s)$-sender-constraint holds because in Step 1, machine a sends to each machine $O(s/c+2)$ words, and thus the total number of words sent is $O(s + c) = O(s)$. The $O(s)$-receiver-constraint holds because in Step 3, each machine receives c messages each consists of $O(s/c + 1)$ words. ◀

Given WeakRangeCast as a building block, we can construct RangeCast where the range $[a, b]$ may be arbitrary in size. The protocol basically builds a distribution tree of fanout s such that the sender first distributes to s machines, then the s machines distribute to upto s^2 machines, and so on. The protocol can be described formally below.

RangeCast

Input and output: Same as WeakRangeCast but without any constraint on the range $[a, b]$.

Protocol:

1. (Base case.) Let $c := b - a + 1$. If $c \leq 1$, then there is only one machine a and it outputs I directly. Otherwise $c > 1$, continue with the following steps.
2. Let $r := \min(c, s)$. Run WeakRangeCast on the first r machines (i.e., in the range $[a, a + r - 1]$) to copy I from machine a to machines $[a, a + r - 1]$.
3. The machine a computes a partition $[a_1, b_1], [a_2, b_2], \ldots, [a_r, b_r]$ of the range $[a, b]$ such that the ranges $[a_i, b_i]$ are as even as possible (i.e., for any $i_1, i_2 \in [r]$, it holds that $|(b_{i_1} - a_{i_1}) - (b_{i_2} - a_{i_2})| \leq 1$). The pair (a_i, b_i) is sent from machine a to both machines $a + i - 1$ and a_i; Afterwards, machine $a + i - 1$ sends the received I to machine a_i for each $i \in [r]$.
4. For each $i \in [r]$, the machine a_i performs recursively RangeCast on the received I and the range $[a_i, b_i]$.

▶ **Lemma 29.** *For any $1 \leq a \leq b \leq m$, RangeCast correctly implements range-cast in $O(\frac{\log m}{\log s})$ rounds and satisfies both $O(s)$-sender- and $O(s)$-receiver-constraints, where each machine takes $O(s)$ space and time locally.*

Proof. In Step 3, if $c < s$, then $r = c$ and all c machines receive I in $O(1)$ rounds. Else, the first s machines receive I and then forward I to groups of size $\lceil c/s \rceil$, and it takes at most $O(\log_s m) = O(\frac{\log m}{\log s})$ iterations to divide any problem of size $c \leq m$ to a constant size. The correctness follows directly. The local-machine complexity, $O(s)$-sender and $O(s)$-receiver constraints follow by WeakRangeCast. ◀

Protocol Replicate

We are now ready to describe the Replicate protocol. We will use the following terminology: we use the term *ball* to refer to a machine's entire state consisting of upto s words. Each ball is always tagged the ball's identifier denoted $\mathsf{id} \in [m]$, i.e., which machine's state it represents.

The building block PrefixSum is the following primitive: every machine starts with a number, and machine i would like to learn the sum of machine 1 to machine i's numbers. As described by Goodrich et al. [70], this can be accomplished in $O(1)$ rounds with an s-sender-bounded MPC protocol, and consuming $O(m)$ total communication.

1. Each ball is additionally tagged with its outputting range computed as follows.
   a. Given as input (β_i, c_i), each machine $i \in [m]$ sets $c_i = 0$ iff $\beta_i = 0$. All m machines jointly run the PrefixSum protocol on $(c_i)_{i \in [m]}$. As the result, each machine i gets the prefix sum $p_i = \sum_{j \in [i]} c_j$.
   b. Let $p_0 = 0$. Every machine i calculates $p_{i-1} = p_i - c_i$ locally. For each machine i such that $c_i \geq 1$ (i.e., the range $[p_{i-1} + 1, p_i]$ is non-empty), machine i tags the range $[p_{i-1} + 1, p_i]$ to its ball and then sends the ball to machine $p_{i-1} + 1$.
2. Now each ball i is received by the first machine in its outputting range $[p_{i-1} + 1, p_i]$. To replicate the ball i to all machines in the range, for each ball i and the tagged range $[p_{i-1} + 1, p_i]$, the machine $p_{i-1} + 1$ performs RangeCast on the ball i and the range of machines $[p_{i-1} + 1, p_i]$. This RangeCast is performed simultaneously for all balls and hence all machines. To ensure all machines finish at the same round, every instance of RangeCast is programmed to finish at the $O(\frac{\log m}{\log s})$-th round specified in Lemma 29.
3. For each machine i such that has a ball tagged with a range $[a, b]$, let $j := i - a + 1$. Output the ball and j.

▶ **Lemma 30.** *The MPC protocol Replicate is a correct replicate protocol such that takes $O(1)$ rounds and $O(m \cdot s)$ communication, satisfies $O(s)$-sender- and $O(s)$-receiver-constraints, and each machine locally takes $O(s)$ time and $O(s)$ space.*

Proof. The correctness holds as each ball i is replicated exactly c_i copies and the ranges $[p_{i-1} + 1, p_i]$ are disjoint. The complexities follow directly by PrefixSum [70] and RangeCast (Lemma 29). ◀

C.2 Sender-Bounded Compiler

We will now compile any MPC protocol that respects only the s-receiver-constraint to one that respects both the s-receiver- and s-sender-constraints.

Without loss of generality, we may assume that at the end of the local computation stage of each round, there is a *deterministic* polynomial time algorithm[14] that takes each machine's local state as input, and can write down sequentially in a stream a set of *send instructions* where each send instruction contains an outgoing word to be sent and the destination machine's identifier. Note that the sender may not have space to write down all these instructions since each word sent multiple times need to be duplicated multiple times, taking more than $O(s)$ space. However, if the sender replicates its state to enough helpers, every helper can locally repeat the same computation, and write down the range of at most $O(s)$ instructions it is responsible for implementing.

Sender-bounded compiler

Every communication round of the original MPC is replaced with the following protocol:

1. Invoke an instance of the non-oblivious Replicate algorithm: if machine i is trying to send in total μ_i words[15], it replicates its local state onto $\lceil \mu_i/s \rceil$ machines. At the end of this phase, every machine i' may receive the local state of at most one machine i and if so, it also learns that it will act as the j-th helper for machine i.
2. If a machine i' is the j-th helper for machine i, it uses machine i's state to compute the $((j-1)s+1)$-th send instruction through the $\min(j \cdot s, \mu_i)$-th send instruction.
3. Now, every machines executes all send instructions written down in the previous step.

Note that this sender-bounded compiler may not compile each round of the original MPC to a fixed number of rounds. To obtain a compiler that always emits a protocol with a fixed number of rounds, we can simply pad the resulting protocol to the worst-case number of rounds: if there is no more work to be done, just execute empty rounds that do nothing.

▶ **Theorem 31** (Sender-bounded compiler with a fixed number of rounds)**.** *Assume that $s = N^{\varepsilon}$ and m is upper bounded by a fixed polynomial in N. Given any m-machine MPC protocol Π that completes in R rounds in the worst case and consuming s per-machine space, there is an MPC protocol Π' that computes the same function as Π, consuming $O(R)$ rounds, $O(s)$ per-machine space, and $O(m \cdot s)$ total communication per round, and moreover Π' additionally satisfies the $O(s)$-sender-constraint, and executes for a fixed number of rounds. Note that for well-formedness, both Π and Π' must satisfy the s-receiver-constraint.*

Proof. By Lemma 30, the above compiler replaces each round of the original MPC with $O(1)$ rounds of communication, and moreover the resulting protocol satisfies $O(s)$-sender- and $O(s)$-receiver-constraints. The fixed total rounds is guaranteed due the padding mechanism mentioned above. ◀

D Additional Preliminary: Robust Secret Sharing

We recall the notion of robust secret sharing schemes [88]. Here, we only consider robust secret sharing schemes with threshold $t < m/3$.

[14] In the case that the algorithm for generating send instructions is randomized, we may assume that the randomness is pseudo-randomly generated with a small seed using a cryptographically secure pseudo-random generator. This way, the MPC's outputs are computationally indistinguishable no matter whether true randomness or pseudorandomness is used in determining the send instructions.

[15] This can be determined with polynomial-time computation based on its local state.

▶ **Definition 32** (Robust Secret Sharing). *A* t-out-of-m robust secret sharing scheme *over a message space* $\mathcal{M}$ *and share space* $\mathcal{S}$ *is a tuple* $(\mathsf{Share}, \mathsf{Recons})$ *of algorithms defined as follows:*

- $\mathsf{Share}(\mathsf{msg}) \to (s_1, \ldots, s_m)$*: This is a randomized algorithm that takes as input a message* $\mathsf{msg} \in \mathcal{M}$ *and output a sequence of shares* $s_1, \ldots, s_m \in \mathcal{S}$*.*
- $\mathsf{Recons}(s_1, \ldots, s_m) \to \mathsf{msg}'$*: This is a deterministic algorithm that takes as input* m *shares* $(s_1, \ldots, s_m)$ *with* $s_i \in \mathcal{S} \cup \{\perp\}$ *and outputs a message* $\mathsf{msg}' \in \mathcal{M}$*.*

We require the following properties.

- ***Perfect Privacy****: Any* t *out of* m *shares of a secret give no information on the secret itself. Namely, for any* $\mathsf{msg}, \mathsf{msg}' \in \mathcal{M}$ *and* $S \subset [m]$ *of size* $|S| = t$*, the distributions of* $\mathsf{Share}(\mathsf{msg})_S$ *and* $\mathsf{Share}(\mathsf{msg}')_S$ *are identical. Here,* $\mathsf{Share}(\mathsf{msg})_S$ *denotes the set of shares* $\{s_i\}_{i \in S}$ *generated by* $\mathsf{Share}(\mathsf{msg})$*.*
- ***Robustness****: An adversary modifies up to* t *shares can cause the wrong secret to be recovered with probability at most* δ*. Specifically, for any* $\mathsf{msg} \in \mathcal{M}$*,* $S \subset [m]$ *of size* $|S| = t$ *and (unbounded) adversary* $\mathcal{A}$*,*

$$\Pr[\mathsf{Recons}(\mathsf{Share}(\mathsf{msg})_{[m]\setminus S}, \mathcal{A}(\mathsf{Share}(\mathsf{msg})_S)) \neq \mathsf{msg}] \leq \delta.$$

It is known that Shamir's secret sharing scheme is an efficient t-out-of-m robust secret sharing scheme for $t < m/3$.