Extended abstract of this work published in ITCS 2016

Secure Multiparty Computation with General Interaction Patterns

Shai Halevi[*] Yuval Ishai[†] Abhishek Jain[‡] Eyal Kushilevitz[§] Tal Rabin[¶]

Abstract

We present a unified framework for studying secure multi-party computation (MPC) with *arbitrarily restricted interaction patterns.* Our study generalizes both standard MPC and recent models for MPC with specific restricted interaction patterns (such as a chain or a star), which were studied by Halevi et al. (Crypto 2011), Goldwasser et al. (Eurocrypt 2014), and Beimel et al. (Crypto 2014).

Since restricted interaction patterns cannot always yield full security for MPC, we start by formalizing the notion of "best possible security" for any interaction pattern. We then obtain the following results:

- **Completeness theorem.** We prove that the star interaction pattern is *complete* for the problem of MPC with general interaction patterns.
- **Positive results.** We present both information-theoretic and computationally secure protocols for computing arbitrary functions with general interaction patterns. We also present more efficient protocols for computing *symmetric* functions and for computing arbitrary functions over a *chain.*
- **Negative results.** We give evidence that our information-theoretic protocols for general functions will be hard to substantially improve on.

All of our protocols rely on a correlated randomness setup, which is *necessary* for computing general functions in our setting. In the computational case, we also present a generic procedure to make any correlated randomness setup *reusable*, in the common random string model.

Although most of our information-theoretic protocols have exponential complexity, they may be practical for functions on small domains (e.g., $\{0,1\}^{20}$), where they are concretely faster than their computational counterparts.

Keywords: Secure multiparty computation, Interactions, Obfuscation.

[*]IBM Research. Email: `shaih@alum.mit.edu`
[†]Technion and UCLA. Email: `yuvali@cs.technion.ac.il`
[‡]Johns Hopkins University. Email: `abhishek@cs.jhu.edu`
[§]Technion, Israel. Email: `eyalk@cs.technion.ac.il`
[¶]IBM Research. Email: `talr@us.ibm.com`

Contents

1 Introduction

Secure multiparty computation (MPC) allows n mutually suspicious parties to evaluate a function on their joint inputs in such a manner that no information about their inputs, beyond the output of the computation, is revealed to each other. Since the first general feasibility results for MPC [52, 35, 6, 15], almost all prior work in this area has considered protocols that require *full interaction* between the parties. Such protocols typically proceed in rounds, where in each round each party may send messages to all other parties, thus requiring that all parties remain online throughout the execution of the protocol.

MPC with Restricted Interaction. Full interaction between the parties is often problematic or even infeasible. For instance, physical distances between wireless devices may prevent them from directly communicating with each other, and temporal constraints may restrict their availability to send or receive messages (e.g., due to battery life). Furthermore, efficiency considerations may also motivate a leaner form of communication. Consider, for instance, the goal of computing the majority vote over the inputs of n parties. While this task can be performed using only $(n-1)$ messages if no security is needed, typical MPC protocols with full interaction involve $\Omega(n^2)$ point-to-point messages to compute the same task securely.

Such considerations have motivated the study of MPC protocols with *restricted* interaction between the parties. Halevi, Lindell, and Pinkas [40] were the first to study this problem: they consider an interaction pattern where each of the n parties, in an ordered fashion, sends a single message to a central server, who eventually computes the output of the function. A different interaction pattern, where each party independently (i.e., without any predetermined order) sends a single message to the server, was recently considered by Goldwasser et al. [36] and Beimel et al. [3]. In both cases, the security guarantee is *necessarily* weaker than the standard simulation-based security for MPC (see below for further discussion on security).

The above works constitute two specific examples of restricted interaction patterns. In general, different application scenarios may dictate different interaction patterns. For example, applications involving data aggregation typically use interaction patterns that can be represented as a directed tree. Furthermore, the topology of the communication network (used by an application) may itself limit the choices of interaction patterns: e.g., a communication network without any node with full degree is not consistent with server-centered interaction patterns. Importantly, as we discuss below, the security guarantees that we get would typically depend on the interaction pattern at hand.

Our Goal: MPC with General Interaction Patterns. Seeking to understand the fundamental role of interaction patterns in MPC, we ask the following broad question:

> Given an arbitrary n-party interaction pattern $\mathcal{I}$ and an n-input function f, can f be securely realized by a protocol that complies with $\mathcal{I}$? If so, how efficiently and under what assumptions?

Before addressing this question, we should clarify how we model interaction patterns, what we mean by "securely," and what setup assumptions we are willing to make.

Modeling Interaction Patterns. A natural starting approach is to represent an interaction pattern as a directed acyclic graph (DAG), where each node represents a party who expects to receive messages from all of its parents and can then send messages to all of its children, and where the sinks of the graph compute outputs. Two simple examples of DAGs include a *chain,* a simple directed path traversing all nodes, and a *star,* a graph connecting all nodes to a single central node.

The protocols from [40, 39] can be adapted to accommodate a chain-based interaction, whereas the protocols from [36, 3] were designed for the case of a star-based interaction. More general DAG-based interaction patterns naturally arise in "self-forming sensor networks," where multiple sensor nodes form an arbitrary communication graph and then collect and compute on data that is transmitted, using the smallest possible number of messages, to a central base station.

While DAGs are an important special case, general interaction patterns are not necessarily restricted to DAGs. Some other useful patterns include the server-centered interaction pattern from [40], a *two-way chain*, where messages travel along a chain from P_n to P_1 and back to P_n who computes an output, or the traditional multi-round protocols over a fully-connected point-to-point network. In Section 2 we describe a unified modeling of general interaction patterns.

1.1 Formulating Achievable Security

In the traditional model of MPC, corrupted parties are restricted to learning the outputs of f on just a *single* input $(x_1, \ldots, x_n)$. However, as observed by [40], this property cannot always be achieved in the case of restricted interaction patterns. For instance, in the server-centered interaction pattern of [40], if the server and the last few parties $P_i, \ldots, P_n$ are corrupted, there is nothing to prevent the adversary from learning the value of f on the honest inputs $x_1, \ldots, x_{i-1}$ and *every possible choice* of corrupted inputs $x_i^*, \ldots, x_n^*$.

To define the "best possible security" for a fixed interaction pattern, we use the notions of free and fixed inputs and residual function from [40]. We call the inputs that the adversary can vary the *free inputs*, and the other inputs are the *fixed inputs*. Clearly, all the honest parties' inputs are fixed. Crucially, however, some of the inputs controlled by the adversary can be fixed as well. For example, in the protocols from [40], the only free inputs are those of corrupted parties that send messages *after the last honest party does*; the inputs of all other corrupted parties are fixed.

Extending the model from [40] to our setting of general interaction patterns, we first consider the case where only one party computes an output and call this party the evaluator. The input of a corrupted party P_i is considered fixed if the interaction pattern includes *any message path* that leads from P_i to the evaluator and passes through *some* honest party. The input of a corrupted P_i is free if *all paths* from P_i to the evaluator consist only of other corrupted parties. For example, in a star pattern with a corrupted center, the inputs of all the corrupted parties are free [36, 3]. In contrast, traditional MPC requires the inputs of all corrupted parties to be fixed.

The "best possible security" is defined by the *residual function*, that captures everything that the adversary can learn about the honest parties' by restricting f to the values of all the fixed inputs while allowing arbitrary choices of the free inputs. Some additional subtleties arise in the malicious-adversary model when multiple parties compute outputs; in particular, security in this case generalizes goals such as Byzantine agreement. We discuss some of these issues further in 2, but leave the issue of malicious-security with multiple output nodes to future work.

An important technical point relates to the achievable notion of simulation. Traditional ideal-vs-real definitions of security (cf. [38, 10, 34]) require *efficient simulation*, but it is known that efficient simulation is, in general, impossible in our setting [40, 36]. For example, protocols with efficient simulation for a star (or even chain) interaction pattern imply virtual black-box obfuscation [36], which is known to be impossible in general [2]. To get around these impossibility results, we settle for security with respect to *indistinguishability* or *unbounded simulation*.

Correlated randomness setup. It is not hard to see that without any form of setup, even very simple functions such as majority cannot always be realized with any meaningful notion of security [3]. Perhaps the simplest model to circumvent such impossibility results is the "minimal model" (PSM) from [28] with general *correlated randomness* setup, where the parties have access to a source $(r_1, \ldots, r_n)$ of correlated random strings. When implementing such protocols, the correlated randomness can come from a trusted dealer or generated using an offline MPC protocol that takes place before the inputs are known and before the limitations on interaction are imposed. This clean model is popular both for a theoretical study of MPC and as a platform for practical implementations that exploit the efficiency benefits of offline preprocessing. (See [7, 22, 3, 23] and references therein.)

The correlated randomness setup can either be *reusable* or *non-reusable*. Namely, it can either be akin to a one-time pad, allowing only a single run of the protocol (which is the case for protocols with information-theoretic security), or it can allow polynomial number of runs (which is sometimes possible in the computational setting). A key advantage of the correlated randomness setup model is that we can hope to tolerate any number of corrupted parties even in the information-theoretic setting [45, 43, 7].

Is "Best Possible Security" Good Enough? Though weaker than the standard notion of security for MPC protocols, our notion of "best possible security" is still meaningful in many interesting cases. First, depending on the interaction pattern and the set of corrupted parties, it could be that most corrupted parties are fixed and hence the residual function is quite degenerate (or even all inputs are fixed as in standard MPC). Second, there are functions for which access even to a "large" residual function does not compromise the secrecy of uncorrupted inputs significantly. Examples include symmetric functions (such as majority) where the size of the residual truth table is not very significant (see [3] for a discussion), as well as unlearnable functions where it is computationally hard to figure out the inputs of honest parties even when given oracle access to the residual function.

1.2 Our Results

We give a variety of answers to the main question posed above. In particular we show "low-end" protocols that offer unconditional security and are generally exponential in the input size (except for special function classes and special interaction patterns), as well as "high-end" protocols that (necessarily) use general-purpose obfuscation techniques to achieve polynomial-time solutions for general functions and interaction patterns.Finally, we also show protocols that make simple use of multilinear maps to compute symmetric functions with general interaction patterns. All of our protocols tolerate an arbitrary number of corrupted parties in the static corruption model. Below, when describing our results, we use the phrase $\mathcal{I}$-compliant to denote that the interaction pattern in a protocol is consistent with $\mathcal{I}$.

I. Completeness Theorem for Interaction Patterns. In Section 3 we show that the star interaction pattern is *complete* for secure computation with restricted interaction patterns. Specifically, we give an efficient, unconditional reduction from the problem of realizing a function f using a general interaction pattern $\mathcal{I}$ to that of realizing the same f on a star. This transformation requires its own (non-reusable) correlated randomness setup, in addition to the setup for the underlying star-pattern protocol, and yields security against malicious parties essentially for free.

Theorem 1 (Informal). *There exists an efficient transformation $\mathcal{T}$ that, for any n-party interaction pattern $\mathcal{I}$ and any star-compliant protocol $\Pi^\star$ for computing a function $f : \{0,1\}^n \to \{0,1\}$, generates an $\mathcal{I}$-compliant protocol $\Pi^{\mathcal{I}}$ for computing f in the non-reusable correlated randomness setup model. If $\Pi^\star$ is statistical/computational semi-honest secure then the resulting $\Pi^{\mathcal{I}}$ is statistical/computational malicious secure. Moreover, the randomness-size (resp. communication com-*

	Correlated Randomness	Online Communication
Previous work: Star [3]	2.5MB	2.5MB
Star (Lemma 4.2)	128KB	128KB
Chain (Theorem 11)	2.5MB	20bits
DAG (Theorem 2)	128KB	1.25MB
Gen. Patterns (Theorem 3)	2.5MB	25MB

Table 1: Concrete complexity numbers (per party) for $n = 20$ for computing general functions with single output with information-theoretic security in the semi-honest model.

plexity) are only a factor of $O(n\lambda)$ (resp. $O(n^2\lambda)$) above those of the underlying protocol $\Pi^\star$, where λ is a security parameter.

Note that the above theorem is stated for the case where f operates over binary inputs. When the function f computed by $\Pi^{\mathcal{I}}$ accepts larger inputs, our transformation requires $\Pi^\star$ for a different (but related) function f' that operates over binary inputs. We remark that although we only consider communication patterns $\mathcal{I}$ with a single sink, in the semi-honest case this can be trivially extended to allow for multiple sink nodes which can each have a different output.

II. Information-Theoretic Protocols for General Functions. In the information-theoretic setting, we present in Section 4 perfectly (resp., statistically) secure protocols for computing any deterministic function against semi-honest (resp., malicious) adversaries. For $f : \{0,1\}^n \to \{0,1\}$, our semi-honest protocols give each party 2^n bits of correlated randomness and require each party to communicate $O(n \cdot 2^n)$ bits. For malicious security, the correlated randomness and communication are of size $O(\lambda n \cdot 2^n)$, with λ the security parameter. In the special case of a star pattern, our protocols improve over previous protocols from [3] by a factor of n.

Theorem 2 (Informal). *For every function $f : \{0,1\}^n \to \{0,1\}$ and any DAG interaction pattern $\mathcal{I}$, there is a semi-honest, perfectly-secure, $\mathcal{I}$-compliant protocol for f, in which each party gets $2^n + 1$ bits of correlated randomness and sends at most $n \cdot 2^{n-1}$ bits of communication. Also, there is a malicious, statistically-secure, $\mathcal{I}$-compliant protocol for f, in which each party gets $O(\lambda n \cdot 2^n)$ bits of correlated randomness and sends at most $O(\lambda n \cdot 2^n)$ bits of communication.*

For non-DAG patterns, we can use our protocol for star (which is a special case of the above; see Section 4.2) and then apply our reduction to obtain:

Theorem 3 (Informal). *For every function $f : \{0,1\}^n \to \{0,1\}$ and any interaction pattern $\mathcal{I}$, there is a semi-honest, perfectly-secure, $\mathcal{I}$-compliant protocol for f, in which each party gets $n+(n+1)\cdot(2^n+1)$ bits of correlated randomness and sends at most $n^2 \cdot (2^{n-1}+2)$ bits of communication. Also, there is a malicious, statistically-secure, $\mathcal{I}$-compliant protocol for f, in which each party gets $O(\lambda n \cdot 2^n)$ bits of correlated randomness and sends at most $O(\lambda n^2 \cdot 2^n)$ bits of communication.*

We stress again that for small input domains, these protocols could be quite practical, see Table 1 for some concrete numbers for computing general functions with binary inputs, a single output and semi-honest security.

Better Communication Complexity. For the chain interaction pattern, we describe in Section 4.1 protocols for computing arbitrary functions where the total communication complexity is only polynomial in the input size, though the correlated randomness is still exponential.

Theorem 4 (Informal)**.** *For every $f : \{0,1\}^n \to \{0,1\}$, there is a semi-honest, perfectly-secure protocol for f with a chain pattern in which each party gets at most $n \cdot 2^n$ bits of correlated randomness and sends at most n bits of communication. Also, there is a malicious, statistically-secure protocol for f with a chain pattern in which each party gets at most $O(\lambda n \cdot 2^n)$ bits of correlated randomness and sends at most $O(n^2 + \lambda n)$ bits of communication.*

In Section 8, we give evidence that it would be hard to extend this result to more general interaction patterns: concretely, we show that even in a network $\mathcal{N}_n$ consisting of two chains (each of length n) that lead to a common endpoint, a similar protocol would imply a 3-server protocol for information-theoretic PIR [18] with poly-logarithmic communication, which is an unexpected result.

Theorem 5 (Informal)**.** *Assume that, for every $f : \{0,1\}^{2n} \to \{0,1\}$, there exists a semi-honest, statistically ϵ-secure $\mathcal{N}_n$-compliant protocol that computes f with communication complexity $c(n)$. Then, there exists an (interactive, statistical) 3-server PIR protocol, with communication complexity $O(c(\log N) + \log N + \log 1/\epsilon)$, where N is the database size.*

III. Efficient Information-Theoretic Protocols for Symmetric Functions. For symmetric functions, we construct in Section 5 *efficient*, perfectly (resp., statistically) secure protocols over a chain against semi-honest (resp., malicious) adversaries where both the offline and online phases are polynomial in the input size.

Theorem 6 (Informal)**.** *For every symmetric binary function $f : \{0,1\}^n \to \{0,1\}$, there is a semi-honest perfectly-secure protocol for f for the chain network in which each party gets $(n+1)^2$ bits of correlated randomness and sends at most $(n+1)^2$ bits of communication. Also, there is a malicious statistically-secure protocol for f for the chain in which each party gets $O(\lambda n^2)$ bits of correlated randomness and sends at most $O(\lambda n^2)$ bits of communication, where λ is the statistical security parameter. Moreover, both these protocols have efficient simulators.*

IV. Computational Protocols for General Functions from Obfuscation. In the computational setting, we observe in Section 6 that the multi-input functional encryption scheme of Goldwasser et al. [36] already yields a protocol for computing general functions with a star interaction against semi-honest adversaries, based on indistinguishability obfuscation (iO) [2, 31] and one-way functions. Combining their result with Theorem 1, we obtain a malicious-secure protocol for computing general functions with general interaction patterns.

Theorem 7 (Informal)**.** *Assuming iO for general circuits and one-way functions, for every interaction pattern $\mathcal{I}$, there exists an $\mathcal{I}$-compliant protocol for computing any polynomial-time function, that achieves malicious security against any number of corruptions, in the (non-reusable) correlated randomness setup model.*

Making Correlated Randomness Reusable. We also present a generic procedure to transform any non-reusable correlated randomness setup into one that is *reusable*. Our transformation works in the common random string (CRS) model where the size of the CRS grows linearly with the number of uses of the correlated randomness. We note, however, that since the CRS is "public" randomness, it can be easily compressed in the random oracle model.

Our transformation builds on the recent work of [41] and inherits their assumptions of iO and fully homomorphic encryption (FHE). Composing our transformation with Theorem 7, we obtain the following:

Theorem 8 (Informal)**.** *Assuming FHE and iO for general circuits, for every interaction pattern $\mathcal{I}$, there exists an $\mathcal{I}$-compliant protocol for computing any polynomial-time function that achieves malicious security against any number of corruptions. The protocol uses a reusable correlated randomness setup in the CRS model where the size of the CRS grows linearly with the number of uses of the protocol.*

Necessity of iO. We note that general-purpose iO is a *necessary* assumption for the above results. Indeed, for the special case of a star pattern, it was already shown by [36] that a secure protocol for general functions implies general-purpose iO.

V. Computational Protocols for Symmetric Functions from Multilinear Maps. For the case of symmetric functions, we describe in Section 7 a much simpler protocol for general interaction patterns that uses multilinear maps but does not require general-purpose obfuscation. The security of that protocol reduces to a very simple variant of the Multilinear Decisional Diffie-Hellman (MDDH) assumption over multilinear maps [9, 30], which we call the *bookend MDDH* assumption. Unfortunately, in light of recent attacks [16, 19, 17, 46], this assumption (and indeed the security of our protocol) *does not hold* for any of the current multilinear map candidates [30, 20, 33, 21]. We hope that future multilinear map constructions will give rise to efficient implementations of our protocol. In fact, this application of multilinear maps can serve as a useful benchmark for evaluating the security and performance of future candidate constructions.

VI. Implications to standard MPC. We note that our results for MPC with general interaction patterns also have relevance to *standard* MPC over *fully connected* networks. For instance, suppose that there is a cost $c_{i,j}$ associated with sending a message from P_i to P_j. Our results reduce the question of minimizing the total cost of an MPC protocol in this setting to a combinatorial optimization problem. For instance, for standard n-party MPC where only P_1 has an output, our results imply general protocols with only $2n-2$ point-to-point messages (e.g., a chain from P_1 to P_n and back), which can be shown to be optimal.

1.3 Technical Overview

We now give an overview of some of our main results, more details are given in technical sections.

Reduction to Star Pattern. Recall that our goal here is to transform an n-party protocol for computing general functions with a star pattern into another protocol for general interaction patterns $\mathcal{I}$. For simplicity, here we focus only on computing functions with binary inputs and achieving semi-honest security see Section 3 for the general case. It is instructive to begin from the naive protocol where each party just sends over the paths in $\mathcal{I}$ to the evaluator whatever message it was supposed to send in the underlying star protocol. This protocol falls short of providing the "best possible security" for the pattern $\mathcal{I}$, because in the star pattern the inputs of all the corrupted parties are free while some of them should be fixed in $\mathcal{I}$.

To do better, we start with the observation that in the underlying star protocol, once all the randomness is fixed, then every party P_i sends one of two fixed messages (m_i^0, m_i^1) to the evaluator, depending upon its input bit. In our transformation, we share the two possible messages of each party using an n-out-of-n secret-sharing, giving each party one share of every message. These shares comprise the correlated randomness that each party gets under our transformation. The idea is that omission of any share will prevent the reconstruction of the original message, so as long as only one of the two shares of party P_i's messages is sent by P_j then P_i's input will be fixed, even if P_i is corrupted.

The challenge is to let P_j know which of the two shares to send to the evaluator without revealing P_i's input. To that end, we distribute the shares of P_i's two messages to all the parties in a *random but consistent order*. That is, either they all get first the share of m_i^0 followed by share of m_i^1, or vice-versa. Furthermore, this random permutation bit is given to party P_i. During the protocol P_i will xor that bit with its input, sending the resulting bit to the other parties to indicate which shares they should send to the evaluator and which to omit. See Section 3 for more details of this protocol and its proof of security.

Information-Theoretic Protocols for General Functions. Our information-theoretic protocols follow a somewhat similar approach to the general reduction discussed above. Specifically, we secret-share the truth table of f, giving each party a share for each of the 2^n inputs, and then have the honest parties omit some of their shares during the protocol run. This is done so as to ensure that when the input of P_i is fixed, the adversary can obtain all the shares for inputs with $x_i = 0$ or all the shares for inputs with $x_i = 1$, *but not both.*

We remark that this is easy to implement in a star pattern: since the only fixed inputs are those of the honest parties, we can have each honest party send to the evaluator only the shares consistent with its own input bit. A protocol for a general pattern can be obtained using the transformation from above, but we can get a better communication complexity by instead tailoring the share-omission rules to the communication pattern. To obtain security in the malicious-adversary model, we add authentication information to the correlated randomness. See Sections 4.1 and 4.2 for details.

Efficient Information-Theoretic Protocols for Symmetric Functions. In the case of symmetric functions, we capitalize on the fact that there is a small representation of the truth table of the function. In particular, the residual function in this representation can be obtained from the global truth table by having each party locally drop one of the rows, depending on its input.

A similar approach was considered by [40] in the computational setting. In their case, the rows of the truth table were encrypted using an additively homomorphic scheme so that party P_{i+1} does not learn what row was dropped by party P_i. In our information-theoretic setting, however, such an additively homomorphic scheme is not available so we use a different hiding mechanism. Specifically, we view messages as matrices, letting each party in the protocol multiply its received message by a random matrix (which is given to it in its correlated randomness), then dropping one column and forwarding the result to the next party. The correlated randomness of evaluator consists of the columns of the resulting product matrix, tagged by the function output and permuted randomly. In a run of the protocol, the evaluator will receive one of these columns and will use it to determine the corresponding function output.

Adding security against malicious adversary is harder here than in our other information-theoretic protocols, since parties do more than just forwarding some pre-determined messages that are given to them as part of their correlated randomness. Our solution still uses authentication to force the corrupted parties to send the right messages, but we must ensure that the added authentication information does not leak information on potential messages that are not sent in a particular run of the protocol. To ensure that, we use an authentication mechanism that doubles also as a randomness extractor, namely the "extractor-MAC" construction from [24], which is based on almost-universal hashing [51]. We also need to withhold from the evaluator the columns of the product matrix, replacing them by just the authentication information needed to recognize these columns. See Section 5 for details and proofs of security.

Reusing correlated randomness. We describe a generic transformation from a non-reusable correlated randomness setup $\mathcal{CR}_{\rm nr}$ into a *reusable* one $\mathcal{CR}$, using a (non-reusable) common random string (CRS).

Our starting idea is to use an MPC protocol Π to compute the function F_{cr} that takes small randomness as input and outputs a large number $L = \mathrm{poly}(k)$ of independent instances of $\mathcal{CR}_{\mathrm{nr}}$. Note that such a function F_{cr} is easy to define: on input a short random seed, F_{cr} first expands it into a large pseudorandom string using a PRG (of appropriate stretch) and then uses different "chunks" of the resultant string to compute L independent instances of $\mathcal{CR}_{\mathrm{nr}}$. Now, fix an honest execution of Π and consider view_i, the view of each party i in the execution, consisting of its random tape and the protocol messages. Note that if each $|\mathsf{view}_i|$ was independent of L, then we could simply set $(\mathsf{view}_1, \ldots, \mathsf{view}_n)$ as an instance of the *reusable* correlated randomness setup $\mathcal{CR}$. This is because given view_i, party P_i can locally compute the output of Π, which consists of L instances of $\mathcal{CR}_{\mathrm{nr}}$.

Thus, we have effectively reduced our problem of making any correlated randomness setup reusable to the problem of constructing an MPC protocol $\Pi_{\text{out-ind}}$ that computes n-party functions with "long" outputs where the communication complexity of the protocol as well as the size of randomness of each party is *independent of the function output length.* A moments reflection, however, reveals that such a protocol is *impossible* in the standard model.[1] Instead, our solution will use a long CRS.

Our starting point is a recent work of Hubáček and Wichs [41], who constructs a secure two-party computation protocol where the communication complexity of the protocol is independent of the function output length. However, the size of the randomness of each party does grow with the function output length. We extend their protocol to the multiparty setting. Let $\Pi_{\text{long-rand}}$ denote the resulting protocol. Our key observation then is that the long randomness of the parties in $\Pi_{\text{long-rand}}$ can be "compressed" by using a long public CRS. In particular, we transform $\Pi_{\text{long-rand}}$ into a new protocol $\Pi_{\text{short-rand}}$ using a public random string $\mathsf{CRS} = \mathsf{CRS}_1, \ldots, \mathsf{CRS}_n$ where each CRS_i is as long as the function output length. The randomness of each party P_i in $\Pi_{\text{short-rand}}$ is set to be a short seed r_i. At the start of the protocol, P_i first locally computes a large random string $R_i = \mathsf{PRG}(r_i) \oplus \mathsf{CRS}_i$. It then executes the strategy of the i'th party in $\Pi_{\text{long-rand}}$ using R_i for the rest of the protocol.

Combining the above steps, we obtain our desired MPC protocol. We stress that we are able to bypass the aforementioned impossibility result since we are working in the CRS model, where the size of the CRS grows with the function output length. (But since the CRS is "public" randomness, it can be easily compressed in the random oracle model.)

Computational Protocols for Symmetric Functions. For symmetric functions, we use multilinear maps to construct simple protocols that achieve computational security against semi-honest adversaries. Here we focus on the star pattern; a protocol for general interaction patterns can be obtained by composing the star protocol with our general reduction to star.

Consider an n-level multilinear map where $[x]_i$ denotes an encoding of x at level i. In our star protocol, as part of the correlated randomness, each party P_i is given two level-1 encodings $[a_i]_1, [a_i \times r]_1$ for random and independent elements a_i's and the same random r. Let $A = \prod_{i=1}^{n} a_i$ denote the product of the a_i's. The evaluator is given of all the $n+1$ (level-n) encodings $[b_i]_n = [A \times r^i], i = 0, 1, \ldots n$ in a random order, where each b_i is tagged with the function value $f_i = f(1^i 0^{n-i})$.

In the protocol, each party P_i simply sends the encoding $[a_i]_1$ *or* $[a_i \times r]_1$, depending upon whether its input bit is 0 or 1, respectively. The evaluator multiplies the n encodings that it receives

[1]Consider an execution of $\Pi_{\text{out-ind}}$ for evaluating a PRG with "long" stretch. The view of any party i in the protocol is a "compressed" representation of the long protocol output. This can be used to derive a computational incompressibility argument, similar to several recent works [1, 13, 41].

and then compares the resulting encoding against the b_i's to determine the function output. More details are found in Section 7. As mentioned above, this protocol is unfortunately insecure using current multilinear-map candidates.

1.4 Related Work

There is a very large body of recent work about minimizing interaction in cryptography (mostly concentrating on the number of rounds, rather than the number of messages): whereas classical MPC results show that every cryptographic task can be realized with sufficient interaction, popular recent research topics such as garbling schemes [53, 5, 37], fully homomorphic encryption [32], functional encryption [49, 8, 48], and obfuscation [2, 31], are all about minimizing interaction. Our work can be seen as taking the question of "what can we do with a given type of interaction" to its ultimate level of generality. One can view the previous notions, as well as standard interactive MPC, as special cases of this general problem.

Another large body of work, originating from [25], studies the problem of secure *communication* (or message transmission) in general networks, where only certain pairs of parties can communicate with each other. This goal is trivialized when allowing a correlated randomness set-up, as we do here, and so the challenges that arise in that setting are very different from the ones we face in our work. The same is true for a recent extension of this problem to secret sharing in general networks [50], a task whose feasibility reduces to that of secure message transmission.

Another line of work, originating from earlier works in the context of distributed computing [26], studies the possibility of realizing MPC on sparse networks [29, 14]. These works use specially designed (expander-based) networks to allow MPC in graphs of a small degree. To this end, they also need to relax the traditional goal of MPC by assuming that some honest inputs are being compromised. However, in contrast to the model considered in our work, being compromised there means "known or fixed by the adversary" as opposed to being "free" in the sense considered here. In particular, general solutions for MPC in that model have no consequences for obfuscation.

Finally, Kearns et al. [44] also study secure computation in a model with restricted communication. Their restriction is more liberal than our definition of interaction pattern: each message from P_i to P_j should be computed by a small neighborhood of P_i, P_j (in an undirected network graph). Their positive results provide computational security against a *single* corrupted party, a limitation which they show to be inherent to their model. In contrast, our protocols provide a meaningful notion of security with respect to any number of corrupted parties.

Acknowledgments. We thank Mor Weiss for her useful comments on an earlier version of this manuscript.

This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467. First and fifth authors supported in part by the Defense Advanced Research Projects Agency (DARPA) and Army Research Office (ARO) under Contract No. W911NF-15-C-0236. Second author supported by ERC grant 259426, a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1228984, 1136174, 1118096, and 1065276, and by the DARPA through the ARL under Contract W911NF-15-C-0205. Second and fourth authors supported by ISF grant 1709/14 and BSF grant 2012378. Third author supported in part by DARPA/ARO Safeware Grant # W911NF-15-C-0213. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

2 Preliminaries

Below we formalize our model of secure computation with arbitrary restricted interaction patterns, generalizing previous definitions from [40, 36, 3] that consider specific patterns. Our definitions assume that the communication pattern is fixed a-priori and does not depend on the input of the parties or their randomness. We begin by defining the syntax for specifying a communication pattern $\mathcal{I}$ and a protocol Π that complies with it. In all the definitions below, we let $\mathcal{P} = \{P_1, \dots, P_n\}$ denote a fixed set of parties who would participate in the protocol. When we want to stress the difference between a protocol message as an entity by itself (e.g., "the 3rd message of party P_1") and the content of that message in a specific run of the protocol, we sometime refer to the former as a "message slot" and the latter as the "message content".

To define an N-message interaction pattern for the parties in $\mathcal{P}$, we assign a unique identifier to each message slot. (Without loss of generality, the identifiers are the indices 1 through N.) An interaction pattern is then defined via a set of constraints on these message slots, specifying the sender and receiver of each message, as well as the other messages that it depends on. These constraints are specified by a *message dependency graph*, where the vertices are the message slots and the edges specify the dependencies.

Definition 1 (Interaction pattern). *An N-message* interaction pattern *for the set of parties $\mathcal{P}$ is specified by a* message dependency *directed acyclic labeled graph,*

$$\mathcal{I} = \big([N],\ D,\ L : V \to \mathcal{P} \times (\mathcal{P} \cup \{\mathsf{Out}\})\big).$$

The vertices are the message indices $[N]$, each vertex $i \in [N]$ is labeled by a sender-receiver pair $L(i) = (S_i, R_i)$, with $R_i = \mathsf{Out}$ meaning that this message is output by party S_i rather than sent to another party.

The directed edges in D specify message dependencies, where an edge $i \to j$ means that message j in the protocol may depend on message i. The message-dependency graph must satisfy two requirements:

- *$\mathcal{I}$ is acyclic. We assume without loss of generality that the message indices are given in topological order, so $i < j$ for every $(i \to j) \in D$.*
- *If message j depends on message i, then the sender of message j is the receiver of message i. That is, for every $(i \to j) \in D$, we have $S_j = R_i$ (where $L(i) = (S_i, R_i)$ and $L(j) = (S_j, R_j)$).*

We assume without loss of generality that each party $P \in \mathcal{P}$ has at most one output, namely at most one $i \in [N]$ such that $L(i) = (P, \mathsf{Out})$. For a message $j \in [N]$, we denote its incoming neighborhood, *i.e. all the messages that it depends on, by $\mathsf{DepOn}(j) \overset{\triangle}{=} \{i : (i \to j) \in D\}$.*

An n-party, N-message interaction pattern, is an N-message pattern for $\mathcal{P} = [n]$. To avoid confusion, we usually denote party i by P_i rather than just the index i.

Note that we allow "party cycles" with one message sent from P_i to P_j and a different message sent from P_j to P_i. Such cycles may have one message depends on the other (e.g., P_i waits for a message from P_j and then replies to it), or they may be independent (e.g., P_i, P_j send independent messages to each other in the same communication round). Some examples of communication patterns with party cycles include:

- Clique. Each party sends a single message to each other party (and then computes an output). The message-dependency graph for this pattern has $N = n^2$ vertices, labeled by all pairs $(R,S) \in \mathcal{P} \times (\mathcal{P} \cup \{\mathsf{Out}\})$ with $R \neq S$. The only dependencies are in the output, namely the message labeled by (P, Out) depends on all the messages labeled by (P', P) (for all $P, P' \in \mathcal{P}$).

- Two-way chain. Messages travel from party P_n to party P_1 on a simple path and then back to party P_n on the same path in the reverse direction, and party P_n is the only one computing an output. Here we have a total of $N = 2n - 1$ messages (including the final output).

- Standard MPC: Standard MPC protocols over point-to-point channels consist of m rounds of interaction, where in each round every party sends a message to every other party, depending on messages it received in previous rounds, and parties compute their outputs at the end of the last round. The clique pattern above is a special case of this pattern with a single communication round.

DAG-based patterns. In some cases we consider special interaction patterns without such party cycles and call them "DAG-based" patterns. Two extreme types of DAGs that we consider in this work are a *star*, containing only one message from P_i to P_n, for $i = 1, \ldots, n-1$, and a *chain*, containing messages from P_i to P_{i+1}, for $i = 1, \ldots, n-1$. In both cases, we think of party P_n as the only party that has an output. For notational convenience, in some of our protocols for stars and chains we use $n+1$ parties, where P_{n+1} is an "evaluator" party who has an output but has no input.

2.1 Protocols with Restricted Interaction Patterns

We next define the syntax of an MPC protocol complying with a restricted fixed interaction pattern. Importantly, our model includes general correlated randomness set-up, making protocols with limited interaction much more powerful.

Definition 2 (MPC with fixed interaction: Syntax). *Let $\mathcal{I} = ([N], D, L)$ be an n-party N-message interaction pattern. An n-party protocol complying with $\mathcal{I}$ is specified by a pair of algorithms $\Pi = (\mathsf{Gen}, \mathsf{Msg})$ of the following syntax:*

- Gen *is a randomized sampling algorithm that outputs an n-tuple of correlated random strings $(r_1, \ldots, r_n)$.*

- Msg *is a deterministic algorithm specifying how each message is computed from the messages on which it depends. Concretely, the input of* Msg *consists of the index $i \in [N]$ of a vertex in the dependency graph, the randomness r_{S_i} and input x_{S_i} for the sender S_i corresponding to that vertex, and an assignment of message-content to all the messages that message i depends on, $M : \mathsf{DepOn}(i) \to \{0,1\}^*$. The output of* Msg *is an outgoing message in $\{0,1\}^*$, namely the string that the sender S_i should send to the receiver R_i.*

The execution of such a protocol Π with pattern $\mathcal{I}$ proceeds as follows. During an offline set-up phase, before the inputs are known, Gen is used to generate the correlated randomness $(r_1, \ldots, r_n)$ and distribute r_i to party P_i. In the online phase, on inputs $(x_1, \ldots, x_n)$, the parties repeatedly invoke Msg on vertices (message-slots) in $\mathcal{I}$ to compute the message-content they should send. The execution of Π goes over the message slots in a topological order, where each message is sent after all messages on which it depends have been received. We do not impose any restriction on the order in which messages are sent, other than complying with the depend-on relation as specified by

$\mathcal{I}$. Once all messages (including outputs) are computed, the parties have local outputs $(y_1, \dots, y_n)$, where we use $y_i = \perp$ to indicate that P_i does not have an output.

For a set $T \subset [n]$ of corrupted parties, let view_T denote the entire view of T during the protocol execution. This view includes the inputs x_T, correlated randomness r_T, and messages received by T. (Sent messages and outputs are determined by this information.) The view does not include messages exchanged between honest parties.

Security of a protocol with communication pattern $\mathcal{I}$ requires that for any subset of corrupted parties $T \subseteq \mathcal{P}$, the view view_T reveals as little about the inputs $x_{\bar{T}}$ of honest parties as is possible with the interaction pattern $\mathcal{I}$. As discussed in the introduction, we formulate this notion of "as little as possible" via the notion of fixed vs. free inputs: If parties P_i, P_j are corrupted and no path of messages from P_i to P_j passes through any honest party, then the adversary can learn the output of P_j on every possible value of x_i. However, if there is some honest party on some communication path from P_i to P_j, then having to send a message through that party may be used to "fix" the input of P_i that was used to generate that message, so the adversary can only learn the value of the function on that one input.

Definition 3 (Fixed vs. free inputs)**.** *For an interaction pattern $\mathcal{I}$, parties $P_i, P_j \in \mathcal{P}$ (input and output parties), and a set $T \subseteq \mathcal{P}$ of corrupted parties, we say that P_i has* fixed input *with respect to $\mathcal{I}$, T and P_j if either*

(1) $P_i \notin T$ (the input party is honest), or

(2) there is a directed path in $\mathcal{I}$ starting with some message sent by P_i, ending with some message received by P_j, and containing at least one message sent by some honest party $P_h \notin T$.

We say that P_i has free input *(with respect to $\mathcal{I}, T, P_j$) if $P_i \in T$ and its input is not fixed. We let $\mathsf{Free}(\mathcal{I}, T, P_j) \subseteq T$ denote the set of parties with free inputs, and $\mathsf{Fixed}(\mathcal{I}, T, P_j) = \mathcal{P} - \mathsf{Free}(\mathcal{I}, T, P_j)$ is the complement set of parties with fixed input (all with respect to $\mathcal{I}$, T and P_j).*

To illustrate the above notion, consider the case where only P_n has output, and T is a strict subset of $\mathcal{P}$ with $P_n \in T$. If $\mathcal{I}$ is a chain, then the free inputs are all $P_i \in T$ whose position in the chain is after the last honest party. In the cases of a star and a clique (or 1-round standard MPC), the entire set T is free. In the cases of a two-way chain and standard m-round MPC with $m \geq 2$, all inputs are fixed. We also note that if the output party is honest then all inputs are fixed, regardless of the interaction pattern.

Using the notion of fixed inputs, we can now capture the minimum information available to the adversary by defining a suitable restriction of the function f that the protocol needs to compute.

Definition 4 (Residual function)**.** *For an n-party functionality f, interaction pattern $\mathcal{I}$, corrupted set $T \subset \mathcal{P}$, input $x = (x_1, \dots, x_n)$ and output party $P_j \in \mathcal{P}$, the* residual function *$f_{\mathcal{I},T,x,P_j}$ is the function obtained from f_j by restricting the input variables indexed by $F = \mathsf{Fixed}(\mathcal{I}, T, P_j)$ to their values in x. That is, for input variables $x'_{\bar{F}} = (x'_i)_{i \notin F}$, we define $f_{\mathcal{I},T,x,P_j}(x'_{\bar{F}}) = f_j(x'_1, \dots, x'_n)$, where $x'_i = x_i$ for all $i \in F$.*

We formalize our notion of security in the semi-honest model below. To get around general impossibility results for security with polynomial-time simulation [40, 36, 3], we will allow by default simulators to be unbounded (but will also consider bounded simulation variants). We start by considering perfectly secure protocols.

Definition 5 (Perfect security with semi-honest adversaries)**.** *Let f be a deterministic n-party functionality, $\mathcal{I}$ be an n-party, N-message interaction pattern, and $\Pi = (\mathsf{Gen}, \mathsf{Msg})$ be an n-party protocol complying with $\mathcal{I}$. We say that Π is a* perfectly T-secure protocol for f *in the semi-honest model for a fixed set $T \subseteq \mathcal{P}$ of corrupted parties if the following requirements are met:*

- Correctness: *For every input $x = (x_1, \ldots, x_n)$, the outputs at the end of the protocol execution are always equal to $f(x)$ (namely, with probability 1 over the randomness of Gen).*
- Perfect security: *There is an unbounded simulator Sim that for any input x is given x_T and the truth tables of the residual functions $f_{\mathcal{I},T,x,P_j}$ for all $P_j \in T$, and its output is distributed identically to $\mathsf{view}_T(x)$.*

We say that Π is a secure protocol for f *if it is T-secure for every $T \subseteq \mathcal{P}$.*

As is typically the case for unbounded simulation, one can equivalently formulate the above definitions in terms of indistinguishability.

Definition 6 (Indistinguishability-based security)**.** *We say that a protocol Π is a T-secure protocol for f in the sense of* indistinguishability *against a semi-honest adversary if it meets the correctness requirement and the following security requirement: For every pair of inputs $x = (x_1, \ldots, x_n), x' = (x'_1, \ldots, x'_n)$ such that $x_T = x'_T$ and $f_{\mathcal{I},T,x,P_j} = f_{\mathcal{I},T,x',P_j}$ for all $P_j \in T$, the random variables $\mathsf{view}_T(x)$ and $\mathsf{view}_T(x')$ are identically distributed.*

Claim 2.1 (Equivalence of unbounded simulation and indistinguishability)**.** *Every protocol Π meets the T-security requirement of Definition 5 if and only if it meets the T-security requirement with respect to indistinguishability of Definition 6.*

Proof. Suppose that the indistinguishability requirement is met. Then a simulator Sim, on input x_T and given oracle access to the residual functions $f_{\mathcal{I},T,x,P_j}$ for all $P_j \in T$, can proceed as follows:

- Using a brute-force search on the input space, find an input x' such that $x'_T = x_T$ and $f_{\mathcal{I},T,x,j} = f_{\mathcal{I},T,x',j}$, for all $j \in T$.
- Run the protocol Π on x' and output the view of T.

The output of the simulator is distributed according to $\mathsf{view}_T(x')$, which by the indistinguishability requirement is identical to the distribution $\mathsf{view}_T(x)$.

In the other direction, suppose there is an unbounded simulator Sim as in Definition 5, and let x and x' be inputs such that $x_T = x'_T$ and $f_{\mathcal{I},T,x,j} = f_{\mathcal{I},T,x',j}$, for all $j \in T$. By the definition of Sim, both $\mathsf{view}_T(x)$ and $\mathsf{view}_T(x')$ should be identically distributed to its output and should therefore have the same distribution, as required. $\square$

Statistical and computational security. Definitions 5 and 6 can be modified in a standard way to capture statistical and computational relaxations of security. In the case of statistical ϵ-security, we allow the correctness requirement to fail with probability at most ϵ and settle for the output of Sim being ϵ-close in statistical distance to $\mathsf{view}_T(x)$ (respectively, for $\mathsf{view}_T(x)$ and $\mathsf{view}_T(x')$ to be ϵ-close). In the case of computational (t, ϵ)-security, we further relax the latter requirements by requiring (t, ϵ)-indistinguishability rather than ϵ-statistical-closeness. Equivalence of simulation and indistinguishability notions holds also for these variants, up to a factor 2 loss in ϵ.

Protocol compilers. As is typically the case in the context of MPC protocols, we will be interested not in a single protocol for a single instance of the problem, but rather in a general solution that can handle different functionalities, interaction patterns, and levels of security in a uniform way. Such a general solution is captured by the following notion of a protocol compiler.

Definition 7 (Protocol compiler)**.** *A* protocol compiler *is a polynomial-time algorithm whose inputs consist of the number of parties n and number of messages N, an n-party functionality f (given by some canonical representation such as a boolean circuit or a truth-table), an n-party N-message interaction pattern $\mathcal{I}$, and (for the case of statistical and computational security) a security parameter 1^k.*

The output of the compiler consists of a pair of boolean circuits implementing a protocol $\Pi = (\mathsf{Gen}, \mathsf{Msg})$ as in Definition 2. The compiler can be either perfect, statistical, or computational. In the computational (resp., statistical) case, it is required that for every polynomial $t(k)$ there is a negligible $\epsilon(k)$ such that the protocol Π, output by the compiler when invoked with security parameter k, is $(t(k), \epsilon(k))$-secure (resp., $\epsilon(k)$-secure).

In the rest of the paper, the term "protocol" will implicitly refer to a protocol compiler as above. We will sometimes restrict protocol compilers to handle special types of functionalities (such as boolean, symmetric, or single-output functionalities), special types of representations (such as a truth-table representation) or special types of interaction patterns (such as a chain or a star).

Bounded simulation. One can similarly define a bounded simulation variant of the above definition, in which the protocol compiler should also output an explicit description of a circuit implementing Sim. This effectively restricts the running time of Sim to be polynomial in the description size of f and the security parameter. While this variant of the definition cannot be realized for general circuits, it can be realized for special types of interaction patterns (such as ones forcing all inputs to be fixed), special functionalities (such as symmetric boolean functions) or when using a truth-table representation of f that effectively allows Sim to be exponential in the bit-length of the inputs.

Reusable set-up. The above definition uses a single invocation of the set-up Gen to support a single evaluation of f. Ideally, one would want to use the same set-up to support an arbitrary polynomial number of function evaluations. While this is impossible to achieve in the information-theoretic setting, some of our protocols for the computational setting have a reusable set-up.

2.2 Security in the presence of malicious adversaries

We define security against malicious adversaries via a suitable modification of the standard real-ideal paradigm for MPC. We start by highlighting some subtleties that arise in this setting.

When considering malicious adversaries, who may deviate from the protocol's specification, one needs to consider not only the information obtained by the adversary about the inputs of honest parties, but also the adversary's influence on the outputs of honest parties. Moreover, while in the semi-honest case the content of messages sent during the protocol execution is insensitive to the message scheduling, malicious adversaries may correlate messages they send with messages they receive, and the latter may depend on the scheduling.[2] We will thus need to guarantee that our

[2]In the case of standard MPC in the synchronous setting with static (non-adaptive) corruptions, one can assume without loss of generality that the adversary is *rushing*, namely that in each round it waits until it receives all message

notion of security hold for every (adversarially chosen) admissible message scheduling, as defined in Definition 1.

Additional subtleties arise in the case where there is more than one output. While the traditional notion of MPC requires that the adversary's inputs be chosen independently of the honest parties' inputs, this is not always possible with general interaction patterns. For instance, in the case of the clique pattern from above, a corrupted P_1 may first compute the output (on each possible choice of its input) after receiving messages from all other parties, and then pick its input depending on the information it learned and send messages honestly according to this input. Moreover, it can potentially use a different input for each destination. This generic strategy allows P_1 to correlate its choice of inputs with the information it can obtain about inputs of honest parties. We will thus need to relax the ideal model to accommodate this type of attacks.

Our definition follows the standard framework for simulation-based security definitions of MPC (cf. [10, 11]). Such definitions require that for any adversary attacking the real protocol there is a simulator attacking an ideal protocol, such that no environment can distinguish between the case it is interacting with the adversary attacking the real protocol and the case it is interacting with the simulator attacking the ideal protocol. The ideal protocol employs a special trusted party for computing the functionality, effectively restricting the simulator to picking its inputs independently of the honest parties' inputs and learning only the outputs of corrupted parties.

One can consider both a standalone variant of these definitions [10, 34], in which the environment only communicates inputs in the beginning of the protocol and receives outputs (from the adversary/simulator and the honest parties) in the end of the protocol, or a universally composable (UC) variant [11, 12] in which the environment can interact freely with the adversary and the simulator. One can also consider both an information-theoretic variant in which the adversary can be unbounded and an information-theoretic variant in which the adversary and the simulator should be efficient. Allowing a correlated randomness setup gets around impossibility results for information-theoretic MPC and UC security.

Our definition deviates from the standard MPC framework in the following ways. We allow simulators to be computationally unbounded by default. We augment the real protocol execution by incorporating the set-up phase, which cannot be controlled by the adversary. (The random inputs r_i of corrupted parties are treated by the simulator as incoming messages it must simulate, similarly to a CRS in standard definitions.) The message scheduling in the real protocol is chosen by the adversary, subject to the constraint of being admissible with respect to the given interaction pattern $\mathcal{I}$.

The interaction of the simulator with the ideal protocol is modified as follows. We start with the simpler case of functionalities having a single output. In this case, if the evaluator (i.e., the output party) is honest then the simulator's interaction with the functionality is as in the standard MPC definition. (This implies, in particular, that the adversary should learn nothing about the honest parties' inputs, and its effect on the output is limited to choosing its inputs independently of honest inputs or making the evaluator output an abort symbol $\perp$.) If the evaluator is corrupted, then the simulator is still required to send a single input value to the functionality for each *fixed* corrupted input (as defined in Definition 3), but it is also allowed to "reset" the functionality by changing an existing value of a *free* corrupted input. The functionality sends an updated output value to the simulator (assuming it has all n inputs) upon each such change.

Note that in the case of unbounded simulation, there is no restriction on the number of times

from the honest parties before sending messages on behalf of corrupted parties. In our asynchronous setting, however, the adversary may face several incomparable scheduling options.

in which the functionality is called. Thus the simulator is effectively given a full description of the residual function defined by the honest inputs and the fixed corrupted inputs, where the latter should be produced by the simulator. (The simulator may pick these fixed inputs by running the adversary's algorithm and extracting effective inputs from messages it sends.)

Functions with Multiple Outputs. Finally, we address the general case of a functionality with multiple outputs. As illustrated by the clique example above, in such a case we need to allow the simulator to learn the corrupted parties' outputs before committing to the inputs that determine the honest parties' outputs. Moreover, it may be possible for the adversary to use different inputs when determining the outputs of different honest parties.

This is captured in the following way. The interaction of the simulator with the ideal functionality is divided into a "learning phase," where the simulator tries to gather as much information as it can about inputs of honest parties, and an "influence phase," where the simulator tries to influence the outputs of honest parties. During the learning phase, honest parties send their inputs to the functionality. Then, for each corrupted output $j \in T$, the simulator interacts with f_j as in the single-output case when $j \in T$. (Namely, it can pick a unique input value for each input that is fixed with respect to T and P_j and can pick arbitrarily many input values for the free inputs; note that the set of free inputs may be different for each j.) During the influence phase, the simulator interacts with each f_h, $h \notin T$, by picking a single input for each $j \in T$, and allowing f_h to deliver the correct output to party h with respect to this input. The simulator can also send $\perp$ to f_h, making party h output $\perp$.

3 A Reduction to Star

Below we show that the star interaction pattern is *complete*, in that we describe a reduction from the problem of realizing a function f using an arbitrary interaction pattern $\mathcal{I}$ to that of realizing the same f on a star. Our transformation is information-theoretic and does not require any cryptographic assumptions, but it requires its own non-reusable correlated randomness (in addition to whatever setup is needed for the underlying star-pattern protocol). We prove that if the underlying star protocol is semi-honest secure then so is the resulting general-pattern protocol, achieving possibly even better security. Furthermore, in Section 3.2 we modify the transformation so that the general-pattern protocol is malicious-secure.

The transformation below assumes that the function f depends on all its inputs, that it has only a single party with output (as this is inherent in having a secure star-protocol for f), and that the interaction pattern has at least one message path from every party to the evaluator. We note that in the semi-honest model one can easily extend a solution for the one output case to multiple outputs by just running separate protocols for the different outputs (piggybacking over the same messages of $\mathcal{I}$ as needed).

It is instructive to consider first the "naive transformation" where every party just sends whatever message it was supposed to send in the underlying star protocol to the evaluator over the paths in $\mathcal{I}$. The simulator of the secure star protocol can be tweaked to provide a simulator for the protocol using $\mathcal{I}$. This naive transformation, however, falls short of what we need because we typically need to provide "more security" in $\mathcal{I}$ than what we had in the underlying star protocol. In a star pattern all the corrupted parties' inputs are free, yet in other interaction patterns some of the corrupt parties' inputs can be fixed. The naive protocol inherits the residual function of the star, yet we want a more restricted residual function in $\mathcal{I}$.

We will modify the naive transformation and cause the inputs of a set of corrupted parties to be fixed. These corrupted parties will include all those who have an honest party on any of their $\mathcal{I}$-paths to the evaluator. Once we expand the set of parties whose input is fixed, we will consider this expanded set to be the set of "honest" parties in the star network. This, in return, will also restrict the residual function that we have access to, thus providing improved security.

The main observation that enables our reduction is that in the star pattern, once the correlated randomness is fixed, each party has only two messages that it can send to the evaluator, depending on its input bit. In our transformation, we share these two possible messages of party P_i using an n-out-of-n secret sharing, giving each party in the graph one share of each message. Note that even party P_i does not know the two messages corresponding to its two possible inputs. Thus, the evaluator needs all shares of an input in order to compute the function on this input. We will further modify the construction to enable an honest P_j on a path of a corrupt P_i to send only one of the shares, de facto fixing the input of P_i.

A remaining challenge is how to let P_j know which of the two shares to send while, at the same time, hiding P_i's input bit. For this, the shares of P_i's two messages are given to all the P_j's *in a random but consistent order*. That is, either they all get first the share of message-0 and then of message-1, or they all get first the share of message-1 and then of message-0. Party P_i is told whether the order is flipped or not. This enables party P_i to inform the parties on its paths to the evaluator which share to send without revealing its actual input bit. If P_j is not on any path from P_i to the evaluator then it will not hear which of the two shares to send, and thus it will forward both shares. If P_i is corrupted, each honest party on the path from P_i to the evaluator would send just one of its two shares, so the view of the protocol would include (at most) one of P_i's underlying messages in the star protocol, which would fix P_i's effective input in the underlying protocol.

3.1 The Semi-Honest Transformation

Theorem 9. *There exists an efficient transformation $\mathcal{T}$ such that for any function $f : \{0,1\}^n \to \{0,1\}$ that depends on all its inputs, any n-party interaction pattern $\mathcal{I}$ with a single sink, and any star-compliant protocol $\Pi^\star$ for f, $\mathcal{T}(\Pi^\star, \mathcal{I})$ is an $\mathcal{I}$-compliant protocol $\Pi^{\mathcal{I}}$ for f (with non-reusable correlated randomness) with the following properties:*

- *If in $\Pi^\star$ each party gets at most R bits of correlated randomness and sends at most M bits of communication, then in $\Pi^{\mathcal{I}}$ each party gets at most $R + 2n \cdot M$ bits of correlated randomness and sends at most $n^2(M+1)$ bits of communication.*
- *If $\Pi^\star$ is perfect/statistical/computational semi-honest secure then so is the resulting $\Pi^{\mathcal{I}}$.*

Proof. Let $P_1, P_2, \ldots, P_{n+1}$ be the parties, and assume without loss of generality that P_{n+1} is the evaluator. Let $\Pi^\star = (\mathsf{Gen}^\star, \mathsf{Msg.int}^\star, \mathsf{Msg.eval}^\star)$ be a protocol for computing f on the star, where $\mathsf{Gen}^\star$ generates correlated randomness, $\mathsf{Msg.int}^\star$ is the next message function of the parties, and $\mathsf{Msg.eval}^\star$ is used by the evaluator to compute the output. For any communication pattern $\mathcal{I}$, we construct a protocol $\Pi^{\mathcal{I}} = (\mathsf{Gen}^{\mathcal{I}}, \mathsf{Msg.int}^{\mathcal{I}}, \mathsf{Msg.eval}^{\mathcal{I}})$ for computing f on $\mathcal{I}$ as follows.

Setup. The randomness-generation procedure $\mathsf{Gen}^{\mathcal{I}}$ begins by running $\mathsf{Gen}^\star$ to generate correlated randomness $r_1, \ldots, r_{n+1}$ for the underlying star protocol, and then proceeds as follows:

1. For each party $i \le n$, and for every input bit $\sigma \in \{0,1\}$, compute a message that P_i could send in the underlying star protocol, $m_i^\sigma \leftarrow \mathsf{Msg.int}^\star(\sigma, r_i)$.

2. Compute an n-out-of-n secret sharing of each message m_i^σ. Let $m_{i,1}^\sigma, \ldots, m_{i,n}^\sigma$ denote the n shares of m_i^σ.

3. Choose random permutation bits $b_1, \ldots, b_n$, one for each party $P_i, i \leq n$.

The correlated randomness of each party P_i includes the permutation bit b_i, and all the pairs $(m_{j,i}^{b_j}, m_{j,i}^{1-b_j})$ for every $j \leq n$. Below we denote $(M_{j,i}^0, M_{j,i}^1) = (m_{j,i}^{b_j}, m_{j,i}^{1-b_j})$. The evaluator receives in addition also r_{n+1}.

Messages and Output. On input x_i, each party P_i computes the bit $c_i = x_i \oplus b_i$, that determines which shares of P_i's messages should be used. Then it proceeds as follows:

1. P_i sends the bit c_i on every path to the evaluator in $\mathcal{I}$.

2. Then, for every P_j such that some path from P_j to the evaluator goes through P_i, party P_i waits until it receives the bit c_j and then sends $M_{j,i}^{c_j} = m_{j,i}^{x_i}$ on the path to the evaluator.

3. For every P_j such that *no path from P_j to the evaluator goes through* P_i, party P_i sends both shares on some $\mathcal{I}$-path to the evaluator.

4. In addition, P_i forwards every message that it receives from other parties on some $\mathcal{I}$-path to the evaluator.

After receiving all the messages, the evaluator collects the set of shares $\{M_{i,1}^{c_i}, \ldots, M_{i,n}^{c_i}\}_{i \in [n]}$ and reconstructs all the messages $m_i^{x_i}$ from the corresponding n shares, for every $i \in [n]$. Finally, it runs the evaluator algorithm $\mathsf{Msg.eval}^\star$ on inputs r_{n+1} and $\{m_i^{x_i}\}_{i \in [n]}$ and returns its output.

This completes the description of $\Pi^{\mathcal{I}}$. The correctness of the protocol is easy to verify.

Complexity. To achieve the communication complexity stated in Theorem 9, the parties need to send their messages only once on all paths to the evaluator and forward each message that they receive only once. Specifically each P_i needs to send the bits c_j (either its own or others') toward each $P_{j'}$ downstream only in the first opportunity that it has according to $\mathcal{I}$. Similarly it needs to send shares toward the evaluator (both its own and forwarded) only in the last opportunity that it has according to $\mathcal{I}$. All other $\mathcal{I}$ messages (if any) should be empty. Done this way, the complexity of the resulting $\Pi^{\mathcal{I}}$ depends only on the number of parties n and NOT on the number of messages N in the communication pattern $\mathcal{I}$.

Some further optimization is possible, in that each P_i need not forward all messages from other parties, it can drop messages that it already knows are inconsistent with the c_j's that it saw. This modification changes the (worst-case) complexity stated in Theorem 9 by at most a small constant factor.

Proof of Security. We need to describe a simulator $\mathsf{Sim}^{\mathcal{I}}$ for the resulting protocol $\Pi^{\mathcal{I}}$, using the simulator $\mathsf{Sim}^\star$ of the underlying star-protocol. The simulator, $\mathsf{Sim}^{\mathcal{I}}$, gets the corrupted parties' input and the residual truth table, and needs to produce the correlated randomness of the corrupted parties and the messages of the honest parties. It will utilize the simulator of the star to achieve this goal. As explained above, the residual truth table that $\mathsf{Sim}^{\mathcal{I}}$ gets is more restricted than that of $\mathsf{Sim}^\star$, as in $\mathcal{I}$ some of the corrupt parties' inputs may be fixed. However, we show that this residual function is sufficient to simulate the communications. Let $T \subseteq [n+1]$ be the set of corrupted parties in a run of $\Pi^{\mathcal{I}}$, and partition it into fixed and free parties $T = T_{\mathsf{Fixed}} \cup T_{\mathsf{Free}}$. Denote the set of honest parties by $H = [n+1] \setminus T$. Let $T^* = T_{\mathsf{Free}}$ be the set of corrupted parties used for simulator $\mathsf{Sim}^\star$ and $H^* = T_{\mathsf{Fixed}} \cup H$ be the set of honest parties.

The $\mathcal{I}$-simulator $\mathsf{Sim}^{\mathcal{I}}$ gets the input bits x_i for all $i \in T$ and the residual function $f'(x_{T_{\mathsf{Free}}}) = f(x_{H^*}, x_{T_{\mathsf{Free}}})$. It runs the star-simulator $\mathsf{Sim}^\star$, giving it the input bits x_i for $i \in T_{\mathsf{Free}}$ and the same residual function. $\mathsf{Sim}^\star$ returns the correlated randomness r_i for $i \in T_{\mathsf{Free}}$ and messages m_i for $i \in H^*$. (Recall that $m_i = m_i^{x_i}$ for some $x_i \in \{0,1\}$, but $\mathsf{Sim}^{\mathcal{I}}$ does not know x_i.)

Next $\mathsf{Sim}^{\mathcal{I}}$ computes the corrupted-party messages of the underlying star protocol as $m_i^\sigma \leftarrow \mathsf{Msg.int}^\star(\sigma, r_i)$ for $i \in T_{\mathsf{Free}}, \sigma \in \{0,1\}$, and also chooses random bits $b_1, \ldots, b_n$. It computes n-out-of-n secret sharing of all the m's that it knows, chooses at random shares for the m's that it does not know, and orders the shares as follows:

- For all $i \in T_{\mathsf{Free}}$, $j \in [n]$, the simulator sets $c_i := b_i \oplus x_i$ and then for $\sigma \in \{0,1\}$ it sets $M_{i,j}^\sigma$ to be the j'th share of the message $m_i^{\sigma \oplus b_i}$.
- For all $i \in H^*$, $j \in [n+1]$, the simulator sets $c_i := b_i$ and then it sets $M_{i,j}^{b_i}$ to be the j'th share of the message m_i, and it chooses $M_{i,j}^{1-b_i}$ uniformly at random.

The $\mathcal{I}$-simulator $\mathsf{Sim}^{\mathcal{I}}$ gives every $i \in T$ the correlated randomness b_i and shares $\{(M_{j,i}^0, M_{j,i}^1) : j \in [n]\}$, and if the evaluator is corrupted then $\mathsf{Sim}^{\mathcal{I}}$ gives it also the correlated randomness r_{n+1} of the underlying star protocol. Finally. $\mathsf{Sim}^{\mathcal{I}}$ runs the actual protocol $\Pi^{\mathcal{I}}$ using the shares that it computed and the c_i bits to determine the honest parties' messages in the $\mathcal{I}$-protocol.

We observe that the simulated view is identical/statistially-close/computationally indistinguishable to the real view, depending on the properties of the underlying simulator $\mathsf{Sim}^\star$, Indeed, if the simulated view contains all the shares of some messages m of the underlying protocol, then either m was produced directly by $\mathsf{Sim}^\star$ or it was computed from the correlated randomness r_i produced by $\mathsf{Sim}^\star$. All other shares in the view, as well as the permutation bits b_i, are uniformly random and independent of everything else. $\square$

Handling Functions Over a Large Domain. The transformation above can handle a function with a large range without any change, but its efficiency relies crucially on the domain being small, so that once the correlated randomness is fixed each party has only a small number of messages that it can possibly send in the star protocol, depending on its input. Applying the transformation as-is to a function $f : D^n \to R$ with a large domain D, would increase the complexity of the star protocol for f roughly by a factor of $|D|$.

To do better, we can always represent each input of f as a binary string of length $\ell = \lceil \log |D| \rceil$, and apply the above transformation to a star protocol for the modified function $f' : \{0,1\}^{n\ell} \to R$, defined as:

$$f'(x_1, \ldots, x_{n\ell}) = f\big((x_1, \ldots, x_\ell), \ldots, (x_{(n-1)\ell+1}, \ldots, x_{n\ell})\big).$$

Note that this requires that we view the n-player interaction pattern $\mathcal{I}$ as an $n\ell$-player pattern, which we can always do by introducing dummy communication flows between the virtual players that are implemented by a single real player. More importantly, though, it requires a star protocol for f' rather than a star protocol for f.

3.2 Reduction for Malicious Security

Theorem 10. *There exists an efficient transformation $\mathcal{T}$ such that for any function $f : \{0,1\}^n \to \{0,1\}$ that depend on all its inputs, any n-party interaction pattern $\mathcal{I}$ with a single sink, and any star-compliant protocol $\Pi^\star$ for f, $\mathcal{T}(\Pi^\star, \mathcal{I})$ is an $\mathcal{I}$-compliant protocol $\Pi^{\mathcal{I}}$ for f (with non-reusable correlated randomness) with the following properties:*

- *If in $\Pi^\star$ the evaluator gets R bits of correlated randomness and the other parties send at most M bits of communication each, then in $\Pi^{\mathcal{I}}$ each party gets at most $R + 2n \cdot M \cdot O(\lambda)$ bits of correlated randomness and sends at most $nR + n^2 M \cdot O(\lambda)$ bits of communication, with λ the statistical security parameter.*
- *If the underlying protocol $\Pi^\star$ is statistical/computational semi-honest secure, then the resulting $\Pi^{\mathcal{I}}$ is statistical/computational malicious secure.*

Proof. (sketch) The transformation for malicious security is very similar to the semi-honest transformation from above, except that we also secret-share the evaluator randomness r_{n+1}, and we authenticate all the messages (say, using an information-theoretic two-time MAC, e.g. 3-wise independent hash functions). Specifically, the parties are given the authentication tags for the messages that they may need to send (and they attach these tags to the messages that they actually send), and the evaluator is given the keys to verify these MACs.

The complexity is easy to verify, and the security proof is quite similar to the one from above. The main difference is that before $\mathsf{Sim}^{\mathcal{I}}$ learns the inputs of the fixed corrupted parties, it needs to give all the corrupted parties their correlated randomness and the messages from honest parties (other than the last one). Moreover, $\mathsf{Sim}^{\mathcal{I}}$ needs to extract these inputs from messages that the fixed corrupted parties send.

To do this, $\mathsf{Sim}^{\mathcal{I}}$ chooses the bits b_i and the shares uniformly at random (and the authentication keys and tags are chosen as in the protocol). When a fixed corrupted party P_i sends the bit c_i towards an honest party, $\mathsf{Sim}^{\mathcal{I}}$ extracts the input bit $x_i = c_i \oplus b_i$. Once it has all the input bits x_i for the fixed corrupted parties, $\mathsf{Sim}^{\mathcal{I}}$ gets the residual function and it can then run the semi-honest simulator $\mathsf{Sim}^\star$ for the underlying star protocol (choosing arbitrary inputs for the free corrupted parties). Now $\mathsf{Sim}^{\mathcal{I}}$ learns from $\mathsf{Sim}^\star$ the correlated randomness r_i for corrupted parties and the messages m_i for the honest parties, so it can compute all the relevant m_i's and choose the shares of the last honest party to match these m_i's (and also the randomness r_{n+1} of the evaluator, if it is corrupted). Finally $\mathsf{Sim}^{\mathcal{I}}$ can compute the authentication tags using the keys that it prepared for the evaluator, so it has everything that it needs for the simulation.

Another difference from the semi-honest case is that, when the evaluator is honest, we need to use the authentication tags. Namely, if the evaluator receives a message which is not consistent with the shares that $\mathsf{Sim}^{\mathcal{I}}$ generated, then the simulator aborts, since this is what would happen whp in the protocol itself.

It is not hard to see that this simulation strategy produces a distribution which is statistically close (upto authentication error) to the real-protocol distribution if $\mathsf{Sim}^\star$ is perfect or statistical, and is computationally indistinguishable from the real-protocol distribution if $\mathsf{Sim}^\star$ is computational. □

4 Information-Theoretic Protocols for General Functions

Below, we present information-theoretically secure protocols with one-time setup for computing arbitrary functions. We begin in Section 4.1 with a protocol for computing arbitrary functions on a chain which is communication efficient (but requires exponential randomness). Then, in Section 4.2, we describe a protocol for computing arbitrary deterministic functions on arbitrary DAGs with exponential communication and randomness. We also describe, in Section 4.2.1, a protocol which is insecure as per Definition 5, but has some interesting implications for garbled circuits.

We identify an n-input function $f(x_1, \dots, x_n)$ with a binary decision tree for f. Each input is associated with a level in the tree and the input ordering in the tree is made to respect the topological order in the DAG. That is, if there is a path from party A to B in the DAG, then we put the input of A before the input of B in the decision tree. In particular, for a chain network, the ordering in the tree agrees completely with the linear order of nodes on the chain.

In the protocols, a party associated with level-i in the tree will be given correlated randomness associated with each edge leading from level i to level $i+1$. During the computation of the function it will send some of the information which is associated with the edges that match its input (i.e. left edges if the input is 0 and right-edges if it is 1). This can be visualized as each party marking some of the edges in its layer of the tree. The markings create a single marked path from the root to a leaf. The value of the function will be computed by the evaluator based on the information in this leaf.

In the description below, we use the following notations: We have a height-n decision tree $\mathcal{T}$, with the root at level 0 and the leaves at level n. The left edge of every intermediate node is labeled with 0, and the right edge is labeled with 1. We name each node in the tree by the labels on the path leading to it, so the root is named ϵ, its left- and right-children are 0 and 1, respectively, their children are 00,01,10,11, etc. (In the protocols we will attach labels to nodes that may be different from their names, but it is convenient to have the names fixed.) Party P_i (who gets the i'th input bit x_i) is associated with level i in the tree,[3] and a special party $\mathcal{E}$ (the *evaluator*) associated with the leaves. The evaluator does not have an input, but it is the one who will learn the value of the function.

4.1 Computing Any Function on a Chain

We consider below only a chain network, and describe *communication-efficient* protocols that still require exponential randomness.

Theorem 11. *For every function $f : \{0,1\}^n \to \{0,1\}$, there is a semi-honest, perfectly-secure, chain-compliant protocol for f in which each party gets at most $n \cdot 2^n$ bits of correlated randomness and sends at most n bits of communication.*

Setup and Correlated Randomness. We have parties $P_0, \dots, P_{n-1}, P_n$ who are connected in a chain starting at P_0, where the last party P_n is the evaluator who does not have an input. Party P_i is associated with level i in the decision tree $\mathcal{T}$ for f.

The correlated randomness in the protocol is determined by a set of random permutations, one for every level in the tree. For level i (with 2^i nodes) we select a random permutation $\pi_i : \{0,1\}^i \to \{0,1\}^i$, and assign to each node with name $x \in \{0,1\}^i$ the label $a = \pi_i(x)$.[4] These node-permutations induce 1-1 mappings as follows: for a node named $x \in \{0,1\}^i$, its label $a =$

[3]For ease of description we start the enumeration of the parties at 0 rather than 1.

[4]Some of our techniques have superficial similarity to [47]; in particular, the use of decision trees and permuting the nodes at each level. However, the setting and goals, as well as the technical details, are very different. In particular, they deal with two-party protocols that have unlimited interaction and they cannot provide information-theoretic security (even with correlated randomness).

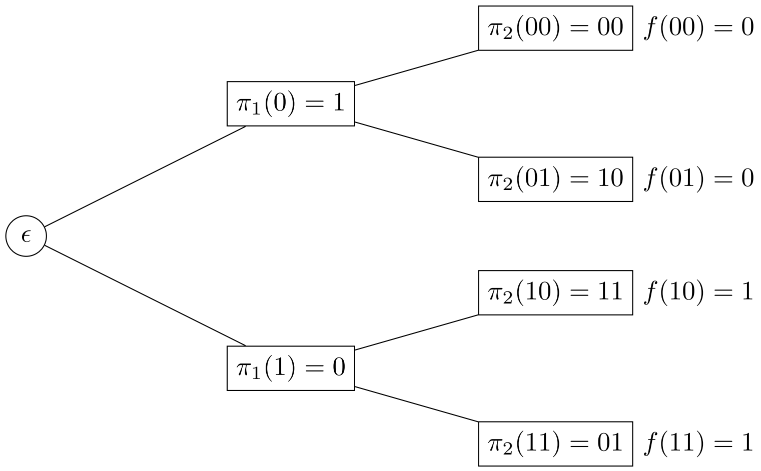

Figure 1: Correlated randomness in the chain protocol. Party i gets the edges from level i to $i+1$, in the form $\{(\pi_i(x), \pi_{i+1}(x|0), \pi_{i+1}(x|1))|x \in \{0,1\}^i\}$, ordered lexicographically by $\pi_i(x)$. The evaluator $\mathcal{E}$ gets the leaves in the form $\{(\pi_n(x) : f(x))|x \in \{0,1\}^n\}$, ordered lexicographically by $\pi_n(x)$.

$\pi_i(x) \in \{0,1\}^i$ and an edge labeled $b \in \{0,1\}$ that leads to its child $(x|b)$, we have the mapping $m_i : \{0,1\}^i \times \{0,1\} \to \{0,1\}^{i+1}$, defined as

$$m_i(a,b) = \pi_{i+1}\big(\pi_i^{-1}(a), b\big).$$

Party P_i is given the above mapping m_i. That is, for the two edges $x \to (x|0), (x|1)$, from any node x at level i to its two children at level i+1, the party P_i is given the tuple $(\pi_i(x), \pi_{i+1}(x|0), \pi_{i+1}(x|1))$. These tuples are given to P_i in an order that does not reveal any extra information about the permutations π_i and π_{i+1} (e.g., in lexicographic order of the $\pi_i(x)$'s). For example, the first party P_0 is given just a single tuple, $(\epsilon, \pi_1(0), \pi_1(1))$, and the second party P_1 is given 2 tuples, $\big((0,\ \pi_2(\pi_1^{-1}(0)|0), \pi_2(\pi_1^{-1}(0)|1))$ and $(1,\ \pi_2(\pi_1^{-1}(1)|0), \pi_2(\pi_1^{-1}(1)|1))\big)$. Overall, P_i is given 2^i such tuples (which takes $(i+1) \cdot 2^{i+1}$ bits to write down).

The evaluator, $\mathcal{E}$ is given the "translation" of π_n to the function values, i.e., $\pi_n(x) \mapsto f(x)$ for all x. In other words, the evaluator is given the table

$$\left\{\langle a : f(\pi_n^{-1}(a))\rangle | a \in \{0,1\}^n\right\},$$

ordered lexicographically by a. An illustration of the permutations and associated function values for a simple 2-input function is given in Figure 1.

Messages in the Chain Protocol. Party P_i with input bit b_i gets a message a_i from its predecessor in the chain P_{i-1} (initially $a_0 = \epsilon$). It applies its mapping to compute $a_{i+1} \leftarrow m_i(a_i, b_i)$ and sends a_{i+1} to the next party P_{i+1}. At the end of the chain, the evaluator $\mathcal{E}$ receives a_n and outputs the corresponding $w_n = f(\pi_n^{-1}(a_n))$ from its table.

Lemma 4.1. *For any n-bit-input function f, the above chain-compliant protocol for computing f is semi-honest secure.*

Proof. Correctness is obvious. For security, let the last honest party be P_t, at level t of the tree. The simulator has the inputs of all the corrupted parties, and the residual truth table of the function, i.e. for all $e_{t+1}, \dots, e_{n-1} \in \{0,1\}^{n-t-1}$ it gets the values $f(b_0, b_1, \dots, b_t, e_{t+1}, \dots, e_{n-1})$ where b_i is the fixed input for $0 \le i \le t$.

The simulator chooses at random a sequence $a_1, \dots, a_{t+1}$. For every party P_i, it fixes a mapping m_i such that $m_i(a_i, b_i) = a_{i+1}$, for all faulty parties, and $m_i(a_i, 0) = a_{i+1}$, for all honest parties. All other values are chosen at random. If P_0 is faulty it fixes $m_0(\epsilon, b_0) = a_1$ and if P_0 is honest $m_0(\epsilon, 0) = a_1$. Furthermore, it chooses random mappings for all faulty parties $P_{t+1}, \dots, P_{n-1}$. The simulator creates the table for $\mathcal{E}$ in the following manner. For $e_{t+1}, \dots, e_{n-1} \in \{0,1\}^{n-t-1}$ it computes the chain $a_{t+2} = m_{t+1}(a_{t+1}, e_{t+1})$ and continues $a_{i+1} = m_i(a_i, e_i)$ until a_n. It sets the value of the table in location a_n to be the value of the function which it received for $e_{t+1}, \dots, e_{n-1}$. The simulator gives the adversary the mappings for the faulty parties, the table for $\mathcal{E}$ and the sequence $a_1, \dots, a_{t+1}$.

In the real execution, if the input of an honest party is 0, then it uses the mapping created by the simulator. If the input is 1, and say that $m_i(a_i, 1) = a'_{i+1}$ then it changes the mapping by switching the two values a_{i+1} and a'_{i+1}. This results in an execution which is identical to the view of the adversary. □

4.1.1 Malicious Security for General Functions on a Chain

We now convert the protocol from Section 4.1 to the malicious-adversary model, simply by authenticating the randomness.

Theorem 12. *For every function $f : \{0,1\}^n \to \{0,1\}$, there is a malicious, statistically-secure, chain-compliant protocol for f in which each party gets $O(\lambda n \cdot 2^n)$ bits of correlated randomness and sends $O(n^2 + \lambda n)$ bits of communication.*

The protocol Recall that in the basic protocol from Section 4.1, the share of correlated randomness given to party i was $\{(\pi_i(x), \pi_{i+1}(x|0), \pi_{i+1}(x|1)) : x \in \{0,1\}^i\}$, ordered lexicographically by $\pi_i(x)$. In the augmented protocol this correlated randomness is augmented by authenticating each of the pairs $(\pi_i(x), \pi_{i+1}(x|0))$ and $(\pi_i(x), \pi_{i+1}(x|1))$. Namely, each of these pairs is authenticated by n different keys for information-theoretic one-time MAC (e.g., pairwise independent hash functions). The authentication tags are given to party i and the keys given to all the other parties, one key per party. Of course different pairs (y, z) are authenticated by different keys. The keys and tags are identified by random unique names, independent of the strings (y, z) themselves, so that parties know what key to use for authenticating what tag.

Note that the total number of keys that a each party receives is n times the number of edges in the tree, so the randomness complexity increases by no more than a factor of $O(n\lambda)$, with λ the statistical security parameter.

In the underlying protocol, party i with input b_i receives a message y_{i-1} from its predecessor in the chain, finds the matching tuple $(y_{i-1}, z_{i,0}, z_{i,1})$ in its share of randomness, sets $y_i = z_{i,b_i}$ and sends y_i to the next party. In the augments protocol, party i will send not only y_i but the pair (y_{i-1}, y_i) with all its authentication tags (one per party downstream). It will also forward the message sent to it by party P_{i-1} (along with all its tags), to let the parties downstream authenticate the various tags. Also, before sending anything, party i will go over all the path so far and use its authentication keys to check that all the transitions $\epsilon \to y_1 \to \cdots \to y_{i-2} \to y_{i-1}$ are properly authenticated.

Any authentication failure, or properly-authenticated inconsistent or malformed messages, will be considered an abort message. Parties will then forward abort messages downstream to cause the entire protocol to be aborted. In particular the evaluator should verify the entire communication that it receives and only output the function value if there are no failures, else outputting a special symbol $\bot$. Note that once any honest party decides to abort, it will not send any authenticated edge $y \to z$, so all honest parties downstream will also abort.

Security. We next describe the simulator for the augmented protocol. It begins by simulating the correlated randomness as follows:

- The simulator chooses a uniformly random permutation ρ_i over $\{0,1\}^i$ for every corrupted party before the last honest party $P_i \in T, i < i^*$, and gives it the edges $\{(z, \rho_i(z0), \rho_i(z1)) : z \in \{0,1\}^i\}$ in lexicographic order over z.
- The simulator next chooses a random permutation π_i over $\{0,1\}^i$ for every $i \geq i^*$, and gives the i'th free corrupted party the edges $\{(\pi_i(x), \pi_{i+1}(x|0), \pi_{i+1}(x|1)) : x \in \{0,1\}^i\}$ in lexicographic order over $\pi_i(x)$, and also gives the evaluator the leaves $\{(\pi_n(x) : f(x)) : x \in \{0,1\}^n\}$ in lexicographic order over $\pi_n(x)$.
- The simulator also chooses authentication keys and their names as in the protocol, gives the corrupted parties their keys and tags, and keeps the keys and tags of the honest parties to itself. Note that even though honest parties $i \notin T$ do not have tuples (y, z) (since their permutations π_i are not yet defined), the simulator generates for each i the appropriate number of keys with random unique names, and gives these keys to all other parties. It will decide what key belongs to what (y, z) tuple later on.

Thereafter, if at any point any corrupted party is able to forge an authentication tag relative to a key that no corrupted party knows, then the simulator aborts. This, of course, happens only with a negligible probability, and that probability accounts for the statistical distance between the real and simulated views. For ease of exposition, we pretend below that this probability is zero.

For each honest party, *except the last one* $i \in \bar{T} \setminus \{i^*\}$, the simulator chooses a uniform random bit-string $y_i \in \{0,1\}^{i+1}$ and uses y_i as the message that this party sends in the underlying protocol from Section 4.1. Using these messages and the authentication keys, the simulator can simulate the execution of the protocol until all honest parties but the last send their messages: Once the incoming message y_{i-1} of an honest party is determined, the simulator decides randomly which of the authentication keys to use for the pair (y_{i-1}, y_i). Then, it computes the authentication tag for that tuple and adds the tag (and its name) to the outgoing message from that party. Note that once the last honest party receives its incoming message, it means that the input of the corrupted parties have been fixed and sent to honest parties. Let us denote also for these parties $i \in T$ the message sent by the i'th party in the underlying protocol by y_i.

If no forgeries occurred and all the authentication checks of the last honest party pass, then in particular all the transitions $y_{i-1} \to y_i$ corresponding to corrupted parties $i \in T$ were taken from the authenticated shared randomness that they received, which means that they correspond to honest protocol message on incoming y_{i-1} and some bit b_i. The simulator then takes these bits b_i to be the fixed inputs of the corrupted parties, and is then given access to the residual function with those fixed corrupted inputs and the honest-party inputs set (and the free corrupted inputs as variables).

The simulator then finds the lexicographic first honest-party input $\tilde{x}$ which is consistent with the residual function and the fixed inputs of the corrupted parties, and for each honest party P_i,

$i < i^*$, it chooses a random permutation π_i subject to the constraint that $\pi_i(y_{i-1}|\tilde{x}_i) = y_i$. We note that this defines also the permutations π_i for the fixed inputs of corrupted parties, as $\pi_i(x|b) = \rho_i(\pi_{i-1}(x)|b)$. Finally, if the last honest party is not the evaluator, then its outgoing message is set to $y_{i^*} = \pi_i(y_{i-1}|\tilde{x}_{i^*})$.

It is clear, by construction, that the simulated view is identical to the adversary's view when the input of the honest parties really was $\tilde{x}$. And, if all the authentication tests pass (and no forgery occurred), then it means that the fixed corrupted parties sent messages that are consistent with the prescribed protocol and the input used by the simulator. In this case, by semi-honest security, the view is also identically distributed to the view on any other honest-parties' input that has the same residual function.

4.2 Protocol for General DAGs

Below, we describe a protocol for computing any deterministic function on any DAG network.

Theorem 13. *For every function $f : \{0,1\}^n \to \{0,1\}$ and any DAG interaction pattern $\mathcal{I}$, there is a semi-honest, $\mathcal{I}$-compliant, perfectly-secure protocol for f, in which each party gets $2^n + 1$ bits of correlated randomness and sends at most $n \cdot 2^{n-1}$ bits of communication.*

To simplify the presentation, we first describe the protocol for a star network where each party just send one message to the evaluator, and later we explain how to generalize. The solution has both communication and randomness complexity that are exponential in the number of parties. In Section 8, we give evidence that achieving a solution with polynomial communication might be impossible.

Recall that we identify a function f with its binary decision tree where, in the case of a star network, the order of inputs is arbitrary and that we denote by P_i the party associated with level i in the tree, and its input is denoted b_i.

4.2.1 A First Attempt

We begin with a protocol which is *insecure* as per Definition 5, but will nonetheless be useful to provide some intuition. In this protocol, every party P_i is given an input-masking random bit r_i and also random output-masking bits s^i_e for every edge e leading from level i to level $i+1$ in the tree. Roughly speaking, the input-masking bits will be used to map the input string $b_0, \ldots, b_{n-1}$ to the root-to-leaf path $b_0 \oplus r_0, \ldots, b_{n-1} \oplus r_{n-1}$ in the tree, and the output-masking bits on the edges of this path will be used to mask the function value that the evaluator gets for that leaf. Namely, the evaluator $\mathcal{E}$ is given a table that holds, for each leaf with name $c_0, \ldots, c_{n-1}$, the value $f(c_0 \oplus r_0, \ldots, c_{n-1} \oplus r_{n-1}) \oplus s^0_{c_0} \oplus s^1_{c_0,c_1} \oplus \cdots \oplus s^{n-1}_{c_0,\ldots,c_{n-1}}$.

During the protocol execution, party P_i computes $c_i \leftarrow b_i \oplus r_i$ and sends to the evaluator the bit c_i and all the output masking bits associated with edges that are consistent with c_i. Namely, the bits $s^i_{e|c_i}$ for all $e \in \{0,1\}^i$. The bits $c_0, \ldots, c_{n-1}$ that the evaluator $\mathcal{E}$ receives define a complete root-to-leaf path in the tree. Moreover the evaluator gets the associated output-masking bits $s^i_{c_0,\ldots,c_i}$ for $i = 0, 1, \ldots, n$, so it can remove the output masking bits to get $f(c_0 \oplus r_0, \ldots, c_{n-1} \oplus r_{n-1}) = f(b_0, \ldots, b_{n-1})$.

While the protocol above is secure when *only the evaluator is corrupted*, it does not satisfy our notion of security. Consider a 3-argument function $f(x_0, x_1, x_2)$, where on input $f(0, x_1, x_2)$ the 4 possible outputs are split to three 0's and a single 1, and for $f(1, x_1, x_2)$ the outputs are balanced

2-2. If the evaluator corrupts P_2 (the party of the last level of the tree) and receives its output-masking bits, it immediately learns whether the right side of the tree corresponds to $f(0, x_1, x_2)$ or to $f(1, x_1, x_2)$. This is due to the fact that the output-masking bits up to that level are the same for the sub tree with the two 0's. Thus, the last output-masking bit reveals these values. When the adversary gets c_0 from P_0, it immediately knows whether its input is 0 or 1.

4.2.2 A Secure Solution

It turns out that the insecure protocol from above can be made secure by using per-leaf output-masking bits rather than per-edge. Namely, each P_i still gets one input-masking bit r_i, but now it gets 2^n output-masking bits, one for each leaf in the tree, $s^i_{c_0,\dots,c_{n-1}}$ for all $(c_0, \dots, c_{n-1}) \in \{0,1\}^n$. The table given to the evaluator also changes accordingly, namely for each leaf with name $c_0, \dots, c_{n-1}$ the table holds the value $f(c_0 \oplus r_0, \dots, c_{n-1} \oplus r_{n-1}) \oplus s^0_{c_0,\dots,c_{n-1}} \oplus s^1_{c_0,\dots,c_{n-1}} \oplus \cdots \oplus s^{n-1}_{c_0,\dots,c_{n-1}}$.

The protocol is modified to reflect the above change as follows. On input b_i, party P_i sets $c_i \leftarrow b_i \oplus r_i$, then it assembles a subset of its output-bit masks that are consistent with c_i; namely, $S_i = \{s^i_{e_0,\dots,e_{i-1},c_i,e_{i+1},\dots,e_{n-1}}\}$ for all $e_0, \dots, e_{i-1}, e_{i+1}, \dots, e_{n-1} \in \{0,1\}^{n-1}$ (in order), and sends (c_i, S_i) to the evaluator. As before, the bits $c_0, \dots, c_{n-1}$ that the evaluator $\mathcal{E}$ receives define a complete root-to-leaf path in the tree, and $\mathcal{E}$ has all the associated output-masking bits $s^i_{c_1,\dots,c_n}$, so it can remove those masks to get $f(c_0 \oplus r_0, \dots, c_{n-1} \oplus r_{n-1}) = f(b_0, \dots, b_{n-1})$. It is easy to see that the different masking for each leaf prevent the adversary from analyzing the tree, yielding the following lemma.

Lemma 4.2. *For any n-bit-input function f, the above star-compliant protocol for function f, is secure in the semi-honest model. Each party gets at most $2^n + 1$ bits of correlated randomness and sends at most $2^{n-1} + 1$ bits.*

Proof. Let $T = \{i_1, \dots, i_l\}$ be the corrupted parties. The simulator has the corrupted-parties' input and the residual function. For the star pattern, all the corrupted parties are free, so the simulator gets all the values $f(b_0, \dots, b_{n-1})$ where the honest parties' input is fixed and the corrupted parties' input vary over all 2^l possibilities.

To compute the table for $\mathcal{E}$, the simulator first chooses random bits $r_0, \dots, r_{n-1}$ and also random bits $s^i_{c_0,\dots,c_{n-1}}$ for all $i \leq n$ and $c_0, \dots, c_{n-1} \in \{0,1\}^n$. It chooses a vector of inputs for the honest parties which is consistent with the inputs of the faulty parties and the residual function; denote by b_i the input chosen for the honest party P_i. For every value of the residual function related to $e_{i_1}, \dots, e_{i_l}$, the simulator computes the leaf to which this value will be assigned as follows: for a fixed input, if the party is honest then it uses the value b_i it chose for it and if the party is faulty it uses the value it got from the adversary, and sets $c_i = b_i \oplus r_i$. For the free inputs, it sets $c_i = e_i \oplus r_i$. It will compute the value for the table at location $c_0, \dots, c_{n-1}$ by taking the value d, from the residual function associated with $e_{i_1}, \dots, e_{i_l}$ and computing $d \oplus s^0_{c_0,\dots,c_{n-1}} \oplus \dots \oplus s^{n-1}_{c_0,\dots,c_{n-1}}$. Once all the values at the locations associated with the residual function have been entered, the simulator completes populating the table for $\mathcal{E}$ with random values.

The simulator computes the view for the adversary by running the computation on the fixed inputs that it had chosen and all the random values. The result of this computation is given to the adversary. It includes all the randomness that relates to the faulty parties (whether their inputs are fixed or free), the table for $\mathcal{E}$ and all the values that the honest parties would transfer to $\mathcal{E}$ during the execution as computed above.

In the real execution for an honest party P_i on input b'_i, the simulator fixes r'_i to be $b_i \oplus r_i \oplus b'_i$. Furthermore, it uses the paddings for the leaves that it chose above. This would result in an identical execution for any input vectors of the honest parties that are consistent with the values of the residual function. □

4.2.3 Extending the Star Protocol to General DAGs

Although we can use the reduction from Section 3 to transform the star protocol to a protocol for any other patterns, below we describe a direct approach that yields more efficient protocols for DAG patterns.

The format of the decision tree and its randomness for the case of the DAG are identical to the star, except that we require the ordering of parties to be consistent with the topological order on the DAG. (Namely each party is assigned to a level of the tree larger than that of all its predecessors in the DAG, and smaller than that of all its successors.) Again, we will call the node assigned to level i party P_i, and give it the input-masking bit r_i and the leaf-masking randomness for each leaf in the tree. Also the evaluator $\mathcal{E}$ is given exactly the same table as in the star protocol above, namely for each leaf with name $c_0, \ldots, c_{n-1}$ the table holds the value $f(c_0 \oplus r_0, \ldots, c_{n-1} \oplus r_{n-1}) \oplus s^0_{c_0,\ldots,c_{n-1}} \oplus \ldots \oplus s^{n-1}_{c_0,\ldots,c_{n-1}}$.

The Protocol. The difference between the star protocol and the general-DAG protocol is that P_i that gets sets of output-masking bits $S^{i'}$ from other parties upstream ($i' < i$) will prune them to be consistent with its input and with each other before forwarding these sets downstream toward the evaluator.[5]

Party P_0 computes $c_0 = b_0 \oplus r_0$, a sequence of labeled output-masking bits

$$S_0 = \left\{ \left\langle (c_0, e_1, \ldots, e_m) : s^0_{c_0,e_1,\ldots,e_m} \right\rangle \mid (e_1, \ldots, e_m) \in \{0,1\}^m \right\}$$

where $m = n - 1$, and a pattern-vector $V_0 = (c_0, *, \ldots, *)$ that reflects the coordinates that P_0 knows to be fixed. As it is at the root of the tree it only knows its own coordinate. P_0 sends $(0, V_0, S_0)$ to all the nodes that it can reach in the DAG.

Party P_i receives a collection of tuples (i_j, V_{i_j}, S_{i_j}) belonging to parties upstream from it, $i_1, \ldots, i_l < i$. (It may get multiple tuples $(i_j, \star, \star)$ for the same i_j on several incoming edges, since this is a semi-honest execution then the corresponding V_{i_j}, S_{i_j} can only differ by having different pruning.) Each vector V_{i_j} reflects inputs that a node upstream knows to be fixed, and hence represents some pruning of previous values.

Party P_i creates the "merged vector" V_i, in which a coordinate k has $V_i[k] = *$ if and only if (a) $V_{i_j}[k] = *$ in *all the incoming vectors* V_{i_j}, and (b) $k \neq i$. It sets the i'th coordinate to be $V_i[i] = c_i$, and the value of every other non-star coordinate is taken from one of the incoming V_{i_j} where this entry is not a $*$. (Again, since this is a semi-honest execution, then if $V_{i_j}[k] \neq *$ it must be the case that $V_{i_j}[k] = c_k$, i.e., this entry contains the c_k value of the party who first set it to a non-$*$ value.)

Then, P_i prunes the sets of output-masking bits in all the S_{i_j}'s to include only those consistent with V_i. In other words, denote the set of labels in S_{i_j} by X_{i_j}, then the set of labels consistent with V_i is the intersection of all the X_{i_j}'s, further reduced to only the labels with the i'th bit equals c_i. Party P_i then drops all the bits from each S_{i_j} that are labeled by inconsistent labels. It

[5]This can be thought of as similar to what happens in the chain protocol, where only a single value needs to be forwarded, but the pruning in this protocol is not as efficient (even if the DAG happens to be a chain).

also computes its own set of labeled output-masking bits

$$S_i = \{\langle (e_0, \ldots, e_m) : s^i_{e_0,\ldots,e_m} \rangle \mid (e_0, \ldots, e_m) \text{ consistent with } V_i\}$$

where $m = n-1$ and sends on all its outgoing edges all the pruned (V_i, S_{i_j})'s together with its own (V_i, S_i).

The evaluator applies exactly the same pruning procedure as before, and if all the nodes are upstream from it then this leaves a single labeled output-masking bit from every party $\langle (c_0, \ldots, c_{n-1}) : s^i_{c_0,\ldots,c_{n-1}} \rangle$. The evaluator looks up the value that it has for the leaf $(c_0, \ldots, c_{n-1})$ and unmask all the output-masking bits to get the function value $f(c_0 \oplus r_0, \ldots, c_{n-1} \oplus r_{n-1}) = f(b_0, \ldots, b_{n-1})$. This completes the description of the protocol.

Lemma 4.3. *For any n-bit-input function f and a DAG pattern $\mathcal{I}$, the above $\mathcal{I}$-compliant protocol for computing the function f is semi-honest secure.*

Proof. The proof for the DAG follows the exact same proof as for the case of the star with the addition that when the simulator computes the execution of the protocol it also creates the vectors V_i and does the same pruning. These vectors can be created in a straightforward manner from all the information which the simulator holds.

A simple change of the randomness will yield an identical execution in the real and simulated worlds. □

4.2.4 Malicious Security for General Functions on a DAGs

As before, we convert the protocol from Section 4.2 to the malicious-adversary model by authenticating the shared randomness.

Theorem 14. *For every function $f : \{0,1\}^n \to \{0,1\}$ and any DAG interaction pattern $\mathcal{I}$, there is a malicious, statistically-secure, $\mathcal{I}$-compliant protocol for f , in which each party gets $O(\lambda n \cdot 2^n)$ bits of correlated randomness and sends $O(\lambda n \cdot 2^n)$ bits of communication.*

The Protocol. Recall that in the semi-honest protocol, the share of correlated randomness for P_i consists of the input-masking bit r_i and the output-masking bits s^i_x for all $x \in \{0,1\}^n$. We note that only $\mathcal{E}$ uses the output-masking bits and thus only he needs to verify their authenticity[6]. However, honest parties in the inner levels of the tree need in some cases to authenticated the masked input bit, $c_i = b_i \oplus r_i$, of other parties. Thus, we augment the correlated randomness by authentication information in the following manner. For each tuple (i, x, s^i_x) we create an information-theoretic one-time MAC (e.g., pairwise independent hash function). The authentication tags are given to P_i and the keys are given to $\mathcal{E}$. In addition, we create the authentication information for the masked input bits, yet the masked input bit are sill unknown. Thus, we will create authenticating information both for the message 0 and 1. The party later will choose which authentication information to use. Here, we create MACS for every two parties P_i, P_j for the messages $(P_i, 0)$ and $(P_i, 1)$. The authentication tag is given to P_i and key is given to P_j.

Note that the total number of keys that each party receives is the number of leaves in the tree plus $2n$ for the masked-input bits. Thus, the randomness complexity only increases by a factor of $O(\lambda)$, where λ is the statistical security parameter.

[6] If the output is computed by multiple parties then each one needs authentication information.

In the protocol, party P_i attaches to each message that it sends all the tags needed to authenticate both input- and output-masking bits along the path to the evaluator. Every party along the way will carry out the computation that the protocol requires to create the set of messages that it needs to transfer and for each of these messages it will include the tags that it received for it (either from other parties or the tags that it holds for the messages). Each party upon receiving tags that it needs to verify carries out the authentication and if it fails or inconsistent messages are received then the party aborts the protocol. A party who has aborted will forward abort messages downstream to cause the entire protocol to be aborted. Furthermore, the evaluator should verify the entire communication that it receives and only output the function value if there are no failures, else outputting a special symbol $\perp$.

Security. We next describe the simulator for the augmented protocol. It begins by simulating the correlated randomness as follows:

- For each party with input except the last honest party, $i \in [n] \setminus \{i^*\}$, the simulator chooses 2^n random output-masking bits $\{s_z^i : z \in \{0,1\}^n\}$, in order of z. [7]
- For corrupt parties $i \in T$ the simulator chooses an input-masking bit r_i and for honest parties $i \notin T$ the simulator chooses a random masked-input bit c_i.
- The simulator choose for the evaluator an output table made of 2^n random bits, $\{t_x : x \in \{0,1\}^n\}$ (in order of x).
- The simulator chooses the authentication keys as in the protocol and computes the corresponding tags (except on tuples $(i^*, \star, \star)$).

The corrupted parties are given their shares of the randomness, and the simulator keeps to itself the rest of the randomness. It then moves to simulating the protocol messages, up until (but not including) the message from the last honest party. This is done just by running the prescribed protocol to compute the outgoing messages, using for each honest party i the random c_i's in the the role of the masked input bits $b_i \oplus r_i$. Note that once the last honest party receives its incoming message, it means that the fixed corrupted parties sent whatever messages they wanted to honest parties.

If no forgeries occurred and all the authentication checks of all the honest parties pass, then it means that the view of every honest party is consistent with the fixed inputs of the corrupted parties. However, it does not mean that the views of different honest parties are consistent (since a corrupted party i can send a message consistent with $x_i = 0$ on one edge and with $x_i = 1$ on another). The simulator takes the input bits for the fixed corrupted parties from the view of the last honest party, and if there are inputs that should be fixed that are not yet fixed in its view then it takes their value from the view of some other (arbitrary) honest party. With the fixed inputs all defined, the simulator gets access to the residual function with those fixed inputs and the honest-party inputs set (and the free inputs as variables).

The simulator then finds the lexicographic first honest-party input which is consistent with the residual function and the fixed inputs. Below we let $\tilde{x}$ denote the fixed inputs (of both honest and corrupt parties). The simulator chooses the rest of the honest-parties' randomness subject to consistency with the input $\tilde{x}$ and protocol execution so far, which for honest parties $i < i^*$ it means

[7] Recall that the "last honest party" is determined by the assignment of parties to levels in the tree, this ordering is consistent with topological order on the communication DAG, but otherwise it is arbitrary.

just setting $r_i \leftarrow c_i \oplus \tilde{x}_i$. For the last honest party i^*, the simulator chooses an input-masking bit r_i at random, sets the output-masking bits that mask the residual function to be consistent with all the other output-masking bits for these inputs and with the function value. Namely for $x = \tilde{x}|z$ and $r = (r_1, \ldots, r_n)$ being the input-masking bits, the simulator sets $s_x^{i^*} \leftarrow f_{\tilde{x},y}(z) \oplus t_{x \oplus r} \oplus_{i \neq i^*} s_x^i$. For any value of x which is not prefixed by $\tilde{x}$ the simulator chooses $s_x^{i^*}$ at random. Finally the simulator uses the randomness of i^* in conjunction with the input value $\tilde{x}_{i^*}$ to compute the message of the last honest party.

By construction, the simulated view is identical to the adversary view when the input of the honest parties is $\tilde{x}$, so it remains to show that the same holds also for any other honest-parties' input that has the same residual function. Here there are three cases, either one of the honest parties detected authentication error, or no authentication error occurred but the views of some honest parties were inconsistent, or no authentication errors and all the views were consistent.

If authentication errors occurred then the honest partys that detect them does not send any of their output-masking bits s_x^i. Hence the adversary view is missing at least one of the output-masking bits s_x^i for each x, and therefore the bits s_x^i in the adversary view are uniformly random (and so the t_x^i'es if the evaluator is corrupted), independently of the honest parties' input and the function values. This is true even conditioned on the rest of the view (and the rest of the view itself is always independent of the honest parties' input).

If no error occurs (and also no forgery) and the views are inconsistent, then there are two honest parties $i, i' \notin T$ that disagree on the x'es used by some corrupted party $P_j \in T$. This means that party P_i only sends its output-masking bits s_x^i for $x_j = 0$ and party $P_{i'}$ only sends its output-masking bits $s_x^{i'}$ for $x_j = 1$. Again it means that the adversary is missing at least one of the output-masking bits s_x^i for every x, and as before it implies that the view is independent of the honest parties' input and the function values.

If no authentication error (or forgery) occurs and the honest parties' views are consistent, it means that all the corrupted parties behave honestly relative to the fixed input that the simulator uses. In this case, by semi-honest security, the adversary view for $\tilde{x}$ is identical to that of any other input with the same residual function.

5 Information-Theoretic Protocols for Symmetric Functions

Below we describe efficient protocols for some simple functions using a chain communication pattern. These protocols are somewhat similar "in spirit" to the ones from the work of Halevi et al. [40], but the technical details are quite different. In particular, our protocols offer one-time perfect security in the correlated-randomness model, while the ones from [40] offer multiple-use computational security in the public-key model. We mention that Gordon et al. [39] generalized the protocols from [40] in the computational setting to a wider class of functions, but these generalizations do not seem to apply to our information-theoretic setting.

Notation. An n-input function f is *symmetric* if for any n inputs $\vec{x} = (x_1, \ldots, x_n)$ and any permutation over $\{1, \ldots, n\}$ it holds that $f(x_1, \ldots, x_n) = f(x_{\pi(1)}, \ldots, x_{\pi(n)})$. When the inputs are binary, $x_i \in \{0, 1\}$, a symmetric function depends only on the Hamming weight of $\vec{x}$. Thus, the truth table of a binary symmetric function f can be described by just $n + 1$ rows, listing the value of f for inputs with Hamming weight $w = 0, 1, \ldots, n$. Moreover, the truth table for the residual function $f_{x_1}(x_2, \ldots, x_n)$ can be obtained from that of f itself by dropping the first row if $x_1 = 1$ or the last row if $x_1 = 0$. This property was used in [40] to describe an efficient protocol for computing symmetric functions in their model, and we can similarly use it in our model.

5.1 Security Against Semi-Honest Adversaries

Theorem 15. *For every symmetric binary function $f : \{0,1\}^n \to \{0,1\}$, there is a semi-honest perfectly secure protocol for f for the chain network (with efficient simulator), in which each party gets $(n+1)^2$ bits of correlated randomness and sends at most $(n+1)^2$ bits of communication.*

The Protocol. We have a chain of $n+1$ parties, where the first n parties, $P_1, \dots, P_n$, have inputs $x_1, \dots, x_n$, respectively, and the last party (the evaluator) has no input but needs to compute the value $f(x_1, \dots, x_n)$. In the pre-processing stage, each party P_i ($i \in [n]$) gets a random invertible $(n+1)$-by-$(n+1)$ matrix R_i (over any field, say $R_i \in \mathcal{Z}_2^{(n+1)\times(n+1)}$). Denote the product of all these matrices, in reverse order, by $C = R_n \times \cdots \times R_1$. The evaluator gets the columns of the product matrix C in random order, and with each column C_i ($i = 0, \dots, n$) it gets the value of f on Hamming-weight-i inputs (note that since C is also invertible then its columns are all distinct). In formula, for a random permutation π, the evaluator is given the pairs

$$\big\{(C_j, y_j) : \; y_{\pi(i)} = f(1^{i-1}0^{n+1-i}) \text{ for } i = 1, \dots n+1\big\},$$

ordered by $j = \pi(i)$. Later, each party P_i has an input bit x_i, and the protocol runs as follows:

0. Denote $A_0 = I$, the $(n+1) \times (n+1)$ identity matrix.

1. For $i = 1$ to n:

   a. Party P_i gets from its predecessor a matrix $A_{i-1} \in Z_2^{(n+1)\times(n+2-i)}$.

   b. Party P_i removes from A_{i-1}, either the first column if $x_i = 1$ or the last column if $x_i = 0$, thus getting an $(n+1) \times (n+1-i)$ matrix A'_{i-1}

   c. Party P_i multiplies A'_{i-1} on the left by its matrix R_i, and sends to P_{i+1} the resulting $(n+1)$-by-$(n+1-i)$ matrix $A_i = R_i \times A'_{i-1}$ (mod 2).

2. The evaluator gets from party P_n a single column $A_n \in Z_2^{(n+1)\times 1}$. If $A_n = C_j$, for some j, then the evaluator outputs the corresponding value y_j.

Correctness. To see why the protocol works, note that the action of each party P_i can be viewed as multiplying A_{i-1} on the left by R_i and on the right by one of two $(n+2-i) \times (n+1-i)$ matrices: either $M_{i,1}$ that drops the first column or $M_{i,0}$ that drops the last one. Hence we have $A_i = R_i \times A_{i-1} \times M_{i,x_i}$ for all i, which means that

$$A_n \;=\; \Big(\prod_{i=n}^{1} R_i\Big) \times I \times \Big(\prod_{i=1}^{n} M_{i,x_i}\Big) \;=\; C \times \Big(\prod_{i=1}^{n} M_{i,x_i}\Big).$$

Next we observe that since the matrices $M_{i,b}$ just drop columns from the matrix they are multiplying on their left, then the product $I \times \prod_{i=1}^{n} M_{i,x_i}$ drops columns from the identity matrix, thus always obtaining a unit vector $\vec{e_j}$ for some $j = 1, \dots, n+1$. This, in turn, implies that $A_n = C \cdot \vec{e_j}$ for some j or, in other words, A_n is indeed a column of C. Moreover, by definition of the $M_{i,b}$'s, we get the j'th column of C iff the Hamming weight of the vector of x_i's is j, as needed.

5.1.1 Proof of Security

Fix the set of corrupted parties $T \subset [n+1]$, and let $i^* \in [n+1] \setminus T$ be the index of the last honest party. Recall that the fixed corrupted parties are $\{i \in T : i < i^*\}$ while the other corrupted parties are free. Let x, x' be two distinct inputs that agree on the inputs of all the corrupted parties, and induce the same residual function on the inputs of the free corrupted parties. To prove security, we need to show that the view of the adversary is distributed the same when the protocol is run with x and x'. The proof for the case $n+1 \notin T$ (i.e., honest evaluator) is trivial, since the adversary only sees random and independent bits in this case, regardless of the input and the function f. Below, we prove it also for the case of corrupted evaluator, $n+1 \in T$ (which means $i^* \le n$). To that end, we describe a one-to-one transformation $\mathcal{T}$ on the randomness space of the protocol, such that the view of the adversary on input x and randomness $\vec{R}$ is the same as its view on input x' and randomness $\mathcal{T}(\vec{R})$. First, we need a few more notations:

Matrices, sets, and permutations. For $i = 0, 1, \ldots, n$, let f_i be the i'th value in the truth-table of f, namely $f_i = f(0^i 1^{n-1})$, and let $\vec{f} = (f_0, f_1, \ldots, f_n)$ be the $(n+1)$-row vector with these values.

For honets parties $i \notin T$, denote $J_i = \prod_{j=1}^{i} M_{j,x_j}$, and $J'_i = \prod_{j=1}^{i} M_{j,x'_j}$. Since multiplying by the $M_{j,b}$'s is the same as dropping columns, we can alternatively view these J matrices as specifying a subset of columns that were not yet dropped (with both J_i, J'_i having $n+1-i$ columns). Specifically, the matrix J consists of the columns of the identity matrix that are indexed by the set J (in order).

We also denote the complement sets by $\bar{J}_i = [n+1] \setminus J_i$ and $\bar{J}'_i = [n+1] \setminus J'_i$, and similarly identify them with matrices. We usually consider the columns of $\bar{J}_i, \bar{J}'_i$ in order, but for the last honest party we use different ordering for the columns of the matrix $\bar{J}'_{i^*}$, as follows: Consider labeling the columns of the identity matrix by the corresponding values of f, namely each column $\mathbf{e}_j$ is labeled by f_j. Since the residual functions relative to x, x' are identical, this means that the columns of J_{i^*}, J'_{i^*} have the same labels in the same order. We then choose the order of columns in $\bar{J}'_{i^*}$ so that also the columns of $\bar{J}_{i^*}, \bar{J}'_{i^*}$ have the same labels in the same order.[8]

We also denote $K_i = (J_i | \bar{J}_i)$ and $K'_i = (J'_i | \bar{J}'_i)$ for all $i \notin T$ (with columns ordered as above), and note that K_i, K'_i are permutation matrices. For the last honest party, the ordering of the columns that we described above implies that $\vec{f} \times K_{i^*} = \vec{f} \times K'_{i^*}$.

For notational convenience, we define $K_0 = K'_0 \overset{\triangle}{=} I$, and for corrupted parties $i \in T, i \le n$ we define $K_i \overset{\triangle}{=} K_{\tilde{i}}$ and $K'_i \overset{\triangle}{=} K'_{\tilde{i}}$, where $\tilde{i}$ is the last honest party before i (or $\tilde{i} = 0$ if there is no such honest party). This implies, in particular, that $K_n = K_{i^*}$ and $K'_n = K'_{i^*}$, and also that for all $i \in T, i \le n$ we have $K_i = K_{i-1}$ and $K'_i = K'_{i-1}$.

The transformation $\mathcal{T}$. Recall that the randomness of the protocol consists of the matrices R_i that are given to the parties, and the permutation applied to the columns of the product (which we denote by Π, i.e. the evaluator gets $Q = C \times \Pi$).

Denote by D_i, for $i \in [n]$, the product of the R_j's so far, $D_i \overset{\triangle}{=} R_i \times R_{i-1} \times \cdots R_1$. In particular $C = D_n$. For notational convenience, set $D_0 = I$. We define $(R'_1, R'_2, \ldots, R'_n, \Pi') = \mathcal{T}(R_1, R_2, \ldots, R_n, \Pi)$ as follows:

- The R'_i's are set so as to get $D'_i K'_i = D_i \times K_i$ for all i. Namely,

$$\forall i \le n, D'_i \leftarrow D_i \times K_i \times (K'_i)^{-1}, \text{ and then } \forall i \le n, R'_i \leftarrow D'_i \times (D'_{i-1})^{-1}. \qquad (1)$$

[8]To make the choice unique, we use the lexicographically first ordering that satisfies this condition.

- The permutation Π' is set as $K'_n\Pi' = K_n\Pi$, namely $\Pi' \leftarrow K'_n \times K_n^{-1} \times \Pi$.

We now need to show that (1) $\mathcal{T}$ is one-to-one, and (2) the adversary view on input x and randomness $\vec{R}$ is the same as on input x' and randomness $\mathcal{T}(\vec{R})$. It is easy to see that $\mathcal{T}$ is one-to-one, since it is invertible: Each R'_i is obtained by multiplying the corresponding R_i by an invertible matrix, and the same holds also for obtaining Π' from Π.

To see that we get the same view, we need to show that for corrupted parties we have $R'_i = R_i$, that the labeled matrices given to the evaluator are the same, and that the messages sent by the honest parties are the same. We begin with the R_i's. From Eqn. (1), we have for all $i \in T$

$$\begin{aligned} R'_i \;=\; D'_i(D'_{i-1})^{-1} \;&=\; \left(D_iK_i(K'_i)^{-1}\right) \times \left(K'_{i-1}K_{i-1}^{-1}D_{i-1}^{-1}\right) \\ &=\; R_iD_{i-1}K_i(K'_i)^{-1}K'_{i-1}K_{i-1}^{-1}D_{i-1}^{-1} \;=\; R_i, \end{aligned}$$

where the last equality follows from $K'_i = K'_{i-1}$ and $K_i = K_{i-1}$.

Next, consider the labeled matrices of the evaluator. For the matrices themselves, we have

$$Q' \;=\; D'_n \times \Pi' \;=\; (D_nK_n(K'_n)^{-1}) \times (K'_nK_n^{-1}\Pi) \;=\; D_n \times \Pi = Q.$$

For the labels, we have $\vec{f} \times \Pi$ in one case and $\vec{f} \times \Pi' = \vec{f} \times K'_nK_n^{-1}\Pi$ in the other. Recalling that we have $K'_n = K'_{i^*}$ and $K_n = K_{i^*}$, we therefore get

$$\vec{f} \times \Pi' = (\vec{f} \times K'_{i^*} \times K_{i^*}^{-1}) \times \Pi = \vec{f} \times \Pi,$$

where the last equality follows because $\vec{f} \times K'_{i^*} = \vec{f} \times K_{i^*}$.

Finally consider the messages of the honest parties $i \notin T$. By definition of D'_i, we have $D'_i \times (J'_i|\bar{J}'_i) = D_i \times (J_i|\bar{J}_i)$, which implies in particular that $A'_i = D'_i \times J'_i = D_i \times J_i = A_i$.

We have shown that all the quantities that the adversary sees are the same under input x and randomness $\vec{R}$ as under input x' and randomness $\vec{R}' = \mathcal{T}(\vec{R})$. This concludes the proof of Theorem 15. □

5.1.2 Extensions

The above protocol can be extended to handle a slightly larger class of functions. Specifically, all functions that can be described by a polynomial-size truth table, where processing each input x_i corresponds to dropping some number of rows from the table, and where

(a) the indices of the rows to drop in step i depend only on the value of x_i, and

(b) the number of rows dropped in step i does not depend on the input.

For example, it was observed in [40, Sec.4.2] that n-input symmetric functions over the domain Z_c have the above properties, with the truth-table size being $\binom{n+c-1}{n} = O(n^c)$. Also it is easy to see that we can handle any binary function which is symmetric in all but $O(\log n)$ of its inputs.

We note that the same protocol can apply to any binary function, but it will not be efficient anymore. (The truth-table size would be 2^n, and processing each bit is done by dropping half the rows in the table.) The resulting protocol does not have any advantages over the general protocols from the previous section.

Finally, we mention that we do not know how to extend the above protocol to arbitrary read-once branching programs. Although we can describe any read-once branching program by a sequence of matrices $M_{i,0}, M_{i,1}$ similar to above, we no longer have the guarantee that the matrices $M_{i,0}$ and $M_{i,1}$ have the same rank, so randomizing via multiplication by a random R_i does not work. This stands in contrast to the computational setting, where Gordon et al. [39] show how to handle arbitrary read-once BPs (and also sparse polynomials). The main difference lies in the fact that in the computational setting one can use suitable partially-homomorphic encryption schemes, whereas we do not have such notion of encryption in the information-theoretic setting.

5.2 Security Against Malicious Adversaries

Trying to examine the protocol from Section 5.1 in the malicious setting, it is not hard to see that the main concern is malicious parties dropping columns from the middle of their matrix before forwarding it in the protocol. For example, consider computing the parity function where only parties 2,3 are honest. The first party can drop (say) the second column from its matrix rather than the first. This would make the residual function after the first party something different than parity, rather than 1-0-1-0...-1-0, the truth table of the residual function will be 0-0-1-0...-1-0. After parties 2,3 had their turn, the others corrupted parties can distinguish the case where they have two zeros from the case where they have two ones, just by checking if the truth table of the residual function after the 3rd party begins with 0-0 or 1-0.

The "obvious" solution to this problem is to authenticate the messages (matrices) that the parties send to each other. At first glance this looks immediate: Fixing the pre-shared randomness, each party i has exactly $n - i$ different messages that it could receive in an honest execution this protocol, so we can give it some authentication material to verufy these messages. The problem is that the extra authentication material can leak information about the other possible messages that this party could have received (in addition to the one that is actually received in the current execution). For example, consider computing the all-0 function in a setting where only the first party is honest. In the protocol from Section 5.1, conditioned on the view of the corrupted parties, there is a unique matrix which is consistent with $x_1 = 0$, and a different unique matrix which is consistent with $x_1 = 1$. If the corrupted parties are also provided with authentication material, they can compute these two matrices and check which of them passes the authentication.

The solution, therefore, is to ensure that the matrices of the honest parties have high entropy, *even conditioned on the view of the corrupted parties and on the input of the honest parties.* To that end, we need to eliminate some of the information that the protocol from Section 5.1 gives to the parties. Specifically, the evaluator will no longer receive the columns of the product matrix C, instead it will only be given the authentication information needed to verify these columns. Choosing authentication keys that are also strong randomness extractors then ensures that they do not leak information about the high-entropy matrices of the honest parties. In the modified protocol, the messages sent by parties are identical to the semi-honest protocol from above. The only differences are that now the parties will use their authentication material to verify these messages (and abort if the verification fails), and the evaluator uses the authentication information (rather than the protocol message itself) to decide what bit to output.

Theorem 16. *For every symmetric binary function $f : \{0,1\}^n \to \{0,1\}$, there is a malicious statistically secure protocol for f for the chain network (with efficient simulator), in which each party gets $O(\lambda n^2)$ bits of correlated randomness and sends at most $O(\lambda n^2)$ bits of communication, where λ is the statistical security parameter.*

5.2.1 The protocol

In the pre-processing stage, each party P_i ($i \in [n]$) gets a random invertible $(n+1)$-by-$(n+1)$ matrix R_i over a large enough field $R_i \in \mathbb{F}^{(n+1)\times(n+1)}$. For concreteness, say that we have $\mathbb{F} = \mathbb{F}_{2^{2\lambda}}$, where each element is represented by 2λ bits. Denote the product of all these matrices, in reverse order, by $C = R_n \times \cdots \times R_1$.

In addition, each party P_i gets i pairs $\{(s_{i,j}, t_{i,j}) : j = 0, \ldots, i-1\}$ in random order, where the $s_{i,j}$'s are authentication keys and the $t_{i,j}$'s are authentication tags relative to these keys, and these (key,tag) pairs double also as extractor seed and output. Specifically, let $M_{i,j}$ be the message that party P_i receives in an honest execution of the protocol with the matrices R_i above when the Hamming weight of the inputs $(x_1, \ldots, x_{i-1})$ is j, then $t_{i,j} := \mathsf{Ext}(s_{i,j}; M_{i,j})$. The evaluator similarly gets $n+1$ authentication pairs $(s_{n+1,j}, t_{n+1,j})$ (in random order), and with such pair it also gets the corresponding output bit, namely $y_j = f(1^j 0^{n-j})$. (The specific construction that we use for authentication and the properties that we need from it are described later in this section.)

Later, each party P_i has an input bit x_i, and the protocol runs as follows:

0. Denote $A_0 = I$, the $(n+1) \times (n+1)$ identity matrix.

1. For $i = 1$ to n:

   a. Party P_i gets from its predecessor a matrix $A_{i-1} \in \mathbb{F}^{(n+1)\times(n+2-i)}$.
   Party P_i goes over all of its authentication pairs (s,t), looking for one satisfying $t = \mathsf{Ext}(s; A_{i-1})$, and aborting if no such pair was found.

   b. Party P_i removes from A_{i-1}, either the first column if $x_i = 1$ or the last column if $x_i = 0$, thus getting an $(n+1) \times (n+1-i)$ matrix A'_{i-1}

   c. Party P_i multiplies A'_{i-1} on the left by its matrix R_i, and sends to P_{i+1} the resulting $(n+1)$-by-$(n+1-i)$ matrix $A_i = R_i \times A'_{i-1} \pmod 2$.

2. The evaluator gets from party P_n a single column $A_n \in \mathbb{F}^{(n+1)\times 1}$. It goes over all of its authentication triple (s,t,y), looking for one satisfying $t = \mathsf{Ext}(s; A_n)$, and outputting the corresponding bit y.

Corresness of this protocol (upto error $2^{-O(\lambda)}$) is obvious.

The Extractor-MAC construction. For our purposes we use an authentication mechanism that also doubles as strong randomness extractors, with the authentication key doubling as the extractor seed. Specifically we use the "extractor-MAC" construction of Dodis et al. [24], which is based on almost-universal hashing [51], and our security proof uses special properties of this construction and its interaction with linear operation over $\mathbb{F}$.

In more detail, the authentication seed consists of three elements of the field $\mathbb{F}_{2^\lambda}$, $s = (x, a, b)$ with $a \neq 0$. (Note that $\mathbb{F}_{2^\lambda}$ is a subfield of $\mathbb{F} = \mathbb{F}_{2^{2\lambda}}$ that we use for the matrices.) We view a messages A to be authenticated as representing a polynomials q_A over $\mathbb{F}_{2^\lambda}$ (using a specific representation that we describe below), the MAC/extractor is then defined as

$$\mathsf{Ext}((x,a,b); A) = a \cdot q_A(x) + b \qquad \text{(operations over } \mathbb{F}_{2^\lambda}\text{)}.$$

The specific representation that we use to identify messages (which are matrices over $\mathbb{F}_{2^{2\lambda}}$) with polynomials over the smaller field $\mathbb{F}_{2^\lambda}$ is chosen to ensure that $\mathbb{F}_{2^{2\lambda}}$-linear operations on these matrices translate to $\mathbb{F}_{2^\lambda}$-linear operations on the coefficients of the corresponding polynomials. (This allows us to describe an efficient simulator below, which can first choose random pairs $(s_{i,j}, t_{i,j})$ and later sample random matrices that are consistent with these choices.) Namely we view $\mathbb{F}_{2^{2\lambda}}$ as a dimension-2 vector space over $\mathbb{F}_{2^\lambda}$, then collect all the entries of a matrix A over $\mathbb{F}_{2^{2\lambda}}$ and view them as the coefficient vector of a polynomial over $\mathbb{F}_{2^\lambda}$.

5.2.2 Proof of Security

Very roughly, we need to show (a) that the corrupted parties cannot deviate from honest execution without being caught whp; and (b) that the view of the corrupted parties is close to uniform independent random bits conditioned on being consistent with the residual truth table. (Moreover we need (b) to hold regardless of the input of the honest parties and actions of the fixed corrupted ones.) Property (a) is implied by the one-time authentication security of the extractor-MAC construction, while property (b) follows from its extraction property (but the proof takes some care).

The technical crux of the proof is showing that the (key,tag) pairs of the corrupted parties (that come after some honest parties) in the real execution are close to being uniformly random bits, independent of their matrices R_i. Fix the input of the honest parties, and partition the parties into alternating intervals of corrupted and honest parties. That is, all the parties upto $P_{i'_1}$ are corrupted, then $P_{i'_1+1}$ through P_{i_1} are honest, then P_{i_1+1} through $P_{i'_2}$ are corrupted, $P_{i'_2+1}$ through P_{i_2} are honest, etc. In particular the honest parties that receive messages from corrupted parties are $P_{i'_1+1}, P_{i'_2+1}, \ldots$ and the honest parties that send messages to corrupted parties are $P_{i_1}, P_{i_2}, \ldots$.

Fix the matrices R_i of all the corrupted parties, and assume by induction that we also fixed the R_i's of all the honest parties upto but not including P_{i_j}. Hence the (key,tag) values of the next interval of corrupted parties $P_{i_j+1}, P_{i_j+2}, \ldots, P_{i'_{j+1}}$ depend only on the random matrix R_{i_j} of the honest P_{i_j} (and on the random ordering that was chosen for them). Ignoring the ordering, we now use the extraction properties of our extractor-MAC to argue that all these pairs are statistically close to uniform, relying on the fact that R_{i_j} has high entropy.

Claim 5.1. *For any fixed values of the corrupted parties' matrices R_i's, as well as all the matrices of honest parties upto but not including R_{i_j}, the (key,tag) pairs of the corrupted parties $P_{i_j+1}, P_{i_j+2}, \ldots, P_{i'_{j+1}}$, viewed as a function of the random choice of R_{i_j}, are jointly statistically close to uniform, upto distance bounded below $n^2/2^\lambda$.*

Proof. Roughly, we argue that since the tags $t_{x,y}$ are only λ bit long and R_{i_i} has nearly $2\lambda(n+1)^2$ bits of min-entropy, then each (key,tag) pair is obtained by applying the extractor on a source that has very high min-entropy *even conditioned on the values of all the other pairs*, and therefore it is close to being uniform and independent of the other pairs. The exact argument is a little more delicate, since each pair depends only of a subset of the columns of R_{i_j} (e.g., the pairs for the evaluator depend each on just a single column). Still we can order these pairs so that each one depends on a high-entropy source even conditioned on all the previous ones, which is what we need.

Specifically, we order them "diagonally", starting from the pair $(s_{i'_{j+1}-1,i'_{j+1}-1}, t_{i'_{j+1}-1,i'_{j+1}-1})$ that is used by the last corrupted party in this interval to authenticate the message corresponding to the all-1 input. Next in the order are the two pairs $(s_{i'_{j+1}-1,i'_{j+1}-2}, t_{i'_{j+1}-1,i'_{j+1}-2})$ and $(s_{i'_{j+1}-2,i'_{j+1}-2}, t_{i'_{j+1}-2,i'_{j+1}-2})$ that depend on a column that the first does not touch (i.e., the last columns that's dropped by $P_{i'_{j+1}-2}$ in an honest execution with the all-1 input).Next are the three pairs $(s_{i'_{j+1}-1,i'_{j+1}-3}, t_{i'_{j+1}-1,i'_{j+1}-3})$, $(s_{i'_{j+1}-2,i'_{j+1}-3}, t_{i'_{j+1}-2,i'_{j+1}-3})$, $(s_{i'_{j+1}-3,i'_{j+1}-3}, t_{i'_{j+1}-3,i'_{j+1}-3})$, that again depend on a column that all the previous ones did no touch, etc. In general we consider "batches" of pairs of the form

$$\{(s_{i_{j+1}-1,k}, t_{i_{j+1}-1,k}), (s_{i_{j+1}-2,k}, t_{i_{j+1}-2,k}), \ldots, (s_{i',k}, t_{i',k})\}$$

(where $i' = \max(i_j, k)$), and we note that each batch contains at most $i'_{j+1} - i_j \leq n$ pairs and depends on an $(n+1)$-dimensional column that all the previous ones did not touch.

Ordering the pairs within each batch arbitrarily, we get that the first pair in each batch has a source of min-entropy at least $2(n+1)\lambda$ even conditioned on all the previous batches, the second has asource with (conditioned) min-entrpy at least $(2n+1)\lambda$, the next has at least $2n\lambda$, etc. As there are at most n pairs in a batch, the last of them has a source with (conditioned) min-entropy at least $2(n+1)\lambda - (n-1)\lambda = (n+3)\lambda$, even conditioned on all previous ones. Hence each pair is at most $2^{-\lambda}$-far from uniform conditioned on all the others.

As there are less than $\binom{n+1}{2}$ pairs that the corrupted parties see, and each is at most $2^{-\lambda}$-far from uniform conditioned on all the previous ones, we conclude that the joint distribution is at most $\binom{n+1}{2}/2^{\lambda}$ far from uniform. □

Armed with Claim 5.1 we can now describe a simulator and prove that the simulated view is statistically close to the real one. The simulator needs to give the corrupted parties their correlated randomness and messages from the honest parties. It also needs to send to the ideal functionality inputs on behalf of the fixed corrupted parties, then it gets access to the residual truth table of the function and needs to complete the simulation.

The simulator begins by choosing uniformly at random the matrices R_i for all the corrupted parties, and the corrupted parties before the first honest parties are given (key,tag) pairs as in the protocol itself. For the other corrupted parties, the simulator chooses the (key,tag) pairs uniformly at random (subject to $a \neq 0$ in the authentication key), and if the evaluator is corrupted then the simulator attaches as many 0's and 1's to the (key,tag) pairs of the evaluator as there are in the truth table of the function f, in random order.

The simulator then goes over the honest parties in their order on the chain, simulating the messages sent by them one by one. As above, we let $P_{i'_1+1}, P_{i'_2+1}, \ldots$ be the honest parties that receive messages from corrupted parties and $P_{i_1}, P_{i_2}, \ldots$ be the honest parties that send messages to corrupted parties, and we show how to simulate the messages of the honest parties P_{i_j} and how to use the messages received by $P_{i'_j+1}$ to extract the inputs of the corrupted parties preceeding it.

On a high level, the simulator samples the message A_{i_j} sent by P_{i_j} uniformly at random, subject to being consistent with the authentication keys and tags of $P_{i_j+1}, \ldots, P_{i'_{j+1}}$. Specifically, the simulator chooses at random one of the (key,tag) pairs that P_{i_j+1} holds, denote it by (s, t), and records the constraint $\mathsf{Ext}(s; A_i) = t$. If P_{i_j+2} is also corrupted then the simulator chooses at random two of its (key,tag) pairs, denoted $(s', t'), (s'', t'')$, and records the two constraints $\mathsf{Ext}(s'; R_{i_j+1}A_{i_j}|_0) = t'$ and $\mathsf{Ext}(s''; R_{i_j+1}A_{i_j}|_1) = t''$, where $R_{i_j+1}A_{i_j}|_0$, $R_{i_j+1}A_{i_j}|_1$ are the matrices obtaind from $R_{i_j+1}A_{i_j}$ by dropping the first or last column, respectively. In general the simulator chooses k of the (key,tag) pairs of P_{i_j+k} and record the constrains that they match all the valid ways of dropping columns from $(\prod_{k=j}^{1} R_{i_j+k}) \times A_{i_j}$.

Once all these constraints are recorded, the simulator samples the message A_{i_j} of P_{i_j} at random subject to all these constraints. Note that with our particular choice of extractor-MACs and the particular representation of matrices as polynomials, the constraints recorded by the simulator are all linear (over $\mathbb{F}_{2^\lambda}$). Hence the simulator can efficiently sample a random solution if one exists. By Claim 5.1 the (key,tag) pairs of the corrupted parties in the simulation are statistically close to those in the real protocol, and since the constraints are always satisfiable in the real execution (bu construction) then they must also be satisfiable in the simulation with overwhelming probability. (If a solution does not exist then the simulator aborts.)

If P_{i_j} is not the last honest party (i.e. $P_{i'_{j+1}+1}$ is the next honest party), then the simulator consider the message $A_{i'_{j+1}}$ that $P_{i'_{j+1}}$ sends to $P_{i'_{j+1}+1}$. If $A_{i'_{j+1}}$ is *not obtained* from $A' = (\prod_{k=i'_{j+1}}^{i_j+1} R_k) \times A_i$ by dropping columns on the right and the left then the simulator makes all the honest parties starting at $P_{i'_{j+1}+1}$ abort. Otherwise, say that $A_{i'_{j+1}}$ is obtained from A' by dropping u columns on the left and u' columns on the right (with $u + u'$ equals the number of corrupted parties in this interval). The simulator sends to the functionality the input bit 1 on behalf of the first u corrupted parties in the interval and the input bit 0 on behalf of the last u' of them, and then proceeds to simulate the message of the next honest party $P_{i_{j+1}}$ as above.

If the evaluator P_{n+1} is corrupted and P_{i_j} is the last honest party ($i_j \leq n$), then the simulator's choice of which (key,tag) pairs of the evaluator to use for the constraints on sampling A_{i_j} must depend the residual function. Before sampling the message of P_{i_j}, the simulator already sent to the functionality the inputs of all the fixed corrupted parties (as described above), and it gets access to the residual truth table. Then the (key,tag) pairs of P_{n+1} that are chosen to match the message A_i must be attached to the 0/1 bits from the residual truth table (in the right order), so the simulator chooses two separate random subsets of pairs, one from the pairs attached to 0 and the other from the pairs attached to 1.

This completes the description of the simulator, it only remains to prove that the simulated view of the corrupted parties is statistically close to their view in a real execution. The view consists of the R_i matrices, the (key,tag) pairs, and the messages from the honest parties. We begin by observing that the one-time authentication properties of our extractor-MAC construction implies that in the real protocol, whenever a corrupted party sends to an honest party a message which is not consistent with a valid way of dropping columns from the product of the R_i's then all subsequent honest parties will abort with overwhelming probability (as they do in the simulating).

To complete the proof we note that in the R_i's are chosen in the same way in both the real protocol and the simulation, and the difference is that in the real world the honest parties' messages are chosen at random (subject to being full rank, which only entails negligible statistical distance) and the (key,tag) pairs are chosen from some distribution that depends on these messages, whereas in the simulation the (key,tag) pairs are chosen at random and the messages are chosen at random subject to being consistent with these pairs.

By Claim 5.1 the marginal distribution of the pairs is the same in both cases (even conditioned on the R_i's of the corrupted parties), and in both case the distribution of the honest parties' messages conditioned on the (key.ta) pairs is uniform amond all the messages that are consistent with these pairs. Hence the entire view of the corrupted parties has nearly the same distribution in both cases, upto a statistical distance of $O(n^2/2^\lambda)$. □

6 Computational Protocols for General Functions

In this section, we describe computationally-secure protocols for computing arbitrary polynomial-time functions with general interaction patterns. Our protocols are built using indistinguishability obfuscation (iO). We stress that for the case of general functions, the use of iO is *necessary*. Indeed it was already shown in [36] that any protocol for computing general functions with star pattern implies iO.

We present our protocols for the following two cases:

- *Non-reusable Correlated Randomness Setup:* We first consider the case where the correlated randomness setup is non-reusable; i.e., it can only be used for a single computation. In this case, we are able to obtain a protocol for general interaction patterns by simply composing the star protocol of [36] with our reduction to star described in Section 3.

- *Reusable Correlated Randomness:* Next, we consider the case where the correlated randomness setup is *reusable*. Here, we present a generic way to transform any non-reusable correlated randomness setup into one that is reusable, provided that the parties have access to a "long" *common random string* (CRS). Combining this transformation with our protocol in the non-reusable setup case, we obtain a protocol for general interaction patterns in the CRS model.

Below, we consider the above two cases in Section 6.1 and 6.2, respectively.

6.1 Computing General Functions with Non-Reusable Correlated Randomness

Here we present a computationally-secure protocol $\Pi^{\mathrm{nr}}_{\mathrm{gen}}$ for computing arbitrary polynomial-time functions with general interaction patterns in the non-reusable correlated randomness model.

We first recall that the recent work of Goldwasser et al. [36] on multi-input functional encryption directly yields a protocol $\Pi^{\mathrm{nr}}_{\mathrm{star}}$ for computing general functions with a star interaction pattern. This protocol makes use of a non-reusable correlated randomness setup and achieves indistinguishability security (or equivalently, unbounded simulation security) against semi-honest adversaries based on $i\mathrm{O}$ and one-way functions.

Theorem 17 (Implicit in [36])**.** *Assuming indistinguishability obfuscation for general circuits and one-way functions, there exists a protocol $\Pi^{\mathrm{nr}}_{\mathrm{star}}$ with non-reusable correlated randomness setup for computing any polynomial-time function on a star, achieving semi-honest security against any number of corruptions.*

Protocol $\Pi^{\mathrm{nr}}_{\mathrm{gen}}$. Let $\Pi^{\mathrm{nr}}_{\mathrm{gen}}$ be the protocol obtained by composing $\Pi^{\mathrm{nr}}_{\mathrm{star}}$ with our reduction from general interaction patterns to star described in Section 3. Note that $\Pi^{\mathrm{nr}}_{\mathrm{gen}}$ also requires a non-reusable correlated randomness setup.

Theorem 18. *Assuming indistinguishability obfuscation for general circuits and one-way functions, for every interaction pattern $\mathcal{I}$, there exists an $\mathcal{I}$-compliant protocol for computing any polynomial-time function in the non-reusable correlated randomness setup model that achieves malicious security against any number of corruptions.*

The proof of the theorem immediately follows by combining the security of the reduction in Section 3 with Theorem 17.

6.2 Reusable Correlated Randomness in the CRS Model

In this section, we present a generic procedure to transform any non-reusable correlated randomness setup into one that is *reusable*. Our transformation works in the common random string (CRS) model where the size of the CRS grows linearly with the number of computations. In particular, for every fresh execution of the protocol that uses the correlated randomness setup, we must use a fresh CRS. We note, however, that since the CRS is "public" randomness, it can be easily compressed in the random oracle model.

We start by providing an overview of our transformation and then proceed to give details.

Overview. Suppose we had an MPC protocol $\Pi_{\text{out-ind}}$ for computing n-party functions with "long" outputs where the communication complexity of the protocol as well as the size of randomness of each party is *independent of the function output length.* Given such a protocol, we can generically transform any non-reusable correlated randomness setup for n parties into a *reusable* one via the following two steps:

- Let Samp be a sampler that generates a single instance of the non-reusable correlated randomness setup. Given Samp and a pseudorandom generator (of appropriate stretch), we can easily define a n-party function F_{cr} that computes $\text{poly}(k)$ independent instances of the correlated randomness setup.

- Fix an honest execution of protocol $\Pi_{\text{out-ind}}$ for evaluating the functionality F_{cr}. Let view_i denote the view of party i in the protocol execution, consisting of its local randomness and the messages in the protocol. Now, consider the correlated randomness $(\mathsf{view}_1, \ldots, \mathsf{view}_n)$ where view_i is given to party i. Due to the efficiency properties of $\Pi_{\text{out-ind}}$, we have that $|\mathsf{view}_i|$ is "small". Furthermore, given view_i, party i can locally compute the output of $\Pi_{\text{out-ind}}$. In other words, $(\mathsf{view}_1, \ldots, \mathsf{view}_n)$ is reusable.

Note that the above transformation only requires semi-honest security of $\Pi_{\text{out-ind}}$ since we only use an honest execution of $\Pi_{\text{out-ind}}$. Unfortunately, however, such a protocol is impossible in the standard model. Indeed, if we consider an execution of $\Pi_{\text{out-ind}}$ for evaluating a PRG with long stretch, then the view of any party i represents a "compressed" representation of the long protocol output. This can be used to derive a computational incompressibility argument, similar to several recent works [1, 13, 41].

Towards that end, we start by noting that a recent work of Hubáček and Wichs [41] constructs a semi-honest secure computation protocol for computing functions with "long" outputs, where the communication complexity of the protocol is independent of the function output length. However, the size of the randomness of each party (who receives the output) does grow with the function output length. The security of their protocol relies upon iO and fully-homomorphic encryption (FHE).

We extend their work on the following two fronts to achieve our goal:

- First, we extend their protocol to the multi-party case. That is, we describe a protocol $\Pi_{\text{long-out}}$ for computing multi-party functions with "long" outputs, where the communication complexity of the protocol is independent of the function output length. The size of the randomness of each party receiving the output, however, still grows with the function output length.

- Next, we describe a generic transformation in the CRS model from any semi-honest protocol $\Pi_{\text{long-rand}}$ where the parties use "long" randomness (i.e., proportional to the function output length) into a protocol $\Pi_{\text{short-rand}}$ where the parties use "short" randomness.

Combining the above two steps, we obtain our desired multi-party protocol $\Pi_{\text{out-ind}}$ where the communication complexity of $\Pi_{\text{out-ind}}$ as well as the size of the randomness of each party in $\Pi_{\text{out-ind}}$ is independent of the function output length. We stress that we are able to bypass the aforementioned impossibility result since we are working in the CRS model, where the size of the CRS grows with the function output length. This completes the overview.

We now proceed to give details.

Multiparty Protocol for Computing Long Outputs using Long Randomness. We first describe an extension of the protocol of [41] to the multiparty case. We borrow much of the terminology and notations from [41] and defer the reader to [41] for relevant definitions and formal security proof.

Notations. Let $i\mathsf{O}$ be an indistinguishability obfuscator for general circuits. Let $(\mathsf{FHE.Keygen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval}, \mathsf{FHE.Rerand})$ denote an FHE scheme with re-randomization. By relying on hybrid encryption, we can assume without loss of generality that $c \leftarrow \mathsf{FHE.Enc}_{pk}(x)$ is of size $\ell_{\mathsf{ctx}} = |x| + \text{poly}(k)$. Let $H = (\mathsf{SSB.Gen}, \mathsf{SSB.Hash}, \mathsf{SSB.Open}, \mathsf{SSB.Verify})$ denote a somewhere statistically binding (SSB) hash function. Here, $\mathsf{SSB.Gen}$ denotes a key sampling algorithm, $\mathsf{SSB.Hash}$ denotes the hashing algorithm, $\mathsf{SSB.Open}$ denotes the opening algorithm, and $\mathsf{SSB.Verify}$ denotes the verification algorithm. Let $\Sigma = \{0,1\}^{\ell_{\mathsf{ctx}}+1}$ denote the alphabet of H with corresponding hash size ℓ_{hash} and opening size ℓ_{opn}. We refer the reader to [41] for a formal definition of SSB hash.[9] Finally, let Π denote a standard semi-honest MPC protocol for computing arbitrary functions.

Protocol $\Pi_{\text{long-out}}$. We now describe a protocol $\Pi_{\text{long-out}}$ for computing any n-party function f that takes inputs $\mathsf{inp}_1, \ldots, \mathsf{inp}_n$ and produces a long output $\mathsf{out} = \mathsf{out}_1, \ldots, \mathsf{out}_L$, where each out_i is a bit. The communication complexity of $\Pi_{\text{long-out}}$ is independent of the function output length, i.e., $CC(\Pi_{\text{long-out}}) = \text{poly}(k,n) + n \cdot \ell_{\mathsf{inp}}(k)$, where ℓ_{inp} denotes the input length of each party. However, the size of the randomness of each party P_i in the protocol grows with the output length of f.

Protocol $\Pi_{\text{long-out}}$ proceeds in the following steps:

1. In the first step, the parties run a standard semi-honest MPC protocol Π to compute the following (randomized) "input-less" function F_{gen}: it samples a key pair $(pk, \mathsf{sk}) \leftarrow \mathsf{FHE.Keygen}(1^k)$ of the FHE scheme and a key $\mathsf{hk} \leftarrow \mathsf{SSB.Gen}(1^k)$ of the SSB hash function H. The output of each party i consists of the tuple $(\mathsf{hk}, pk, \mathsf{sk}_i)$ where $\mathsf{sk} = \mathsf{sk}_1 \oplus \cdots \oplus \mathsf{sk}_n$.

2. In the next step, each party i broadcasts an encryption $c_i \leftarrow \mathsf{FHE.Enc}_{pk}(\mathsf{inp}_i)$ of its input inp_i to all the other parties. Let $\vec{C}_{-i}$ denote the ciphertexts received by party i.

3. Next, each party i performs the following sequence of steps:
   - Compute $c^i_{\mathsf{out}} \leftarrow \mathsf{FHE.Eval}_{pk}\left(f(\cdot, \mathsf{inp}_i, \cdot), \vec{C}_{-i}\right)$.
   - Choose a random "pad" z_i and compute $c^i_{\mathsf{pad}} \leftarrow \mathsf{FHE.Eval}_{pk}(\mathsf{OTP}_{z_i}, c^i_{\mathsf{out}})$ where $\mathsf{OTP}_z(x) := x \oplus z$ is the one-time pad function.
   - Re-randomize c^i_{pad} to compute $c^i_{\mathsf{frsh}} \leftarrow \mathsf{FHE.Rerand}_{pk}(c^i_{\mathsf{pad}})$. Let $c^i_{\mathsf{frsh}} = c_i[1], \ldots, c_i[L]$.
   - Choose randomness $a = a_1, \ldots, a_L$. For every $j \in [L]$, let $x_i[j] = (c_i[j], a_j)$. Let $x_i = x_i[1], \ldots, x_i[L]$. Compute $y_i \leftarrow \mathsf{SSB.Hash}_{\mathsf{hk}}(x_i)$.

4. Now, the parties run a standard semi-honest MPC protocol Π to compute the following function F_{dec}: the input of party i is the tuple $(\mathsf{hk}, y_i, \mathsf{sk}_i)$. The output of party i is $\tilde{C}_i \leftarrow i\mathsf{O}(C_i)$ where the circuit $C_i = C_{[\mathsf{hk}, y_i, \mathsf{sk}]}$ contains the SSB hash key hk, the hash value y_i and the FHE secret key $\mathsf{sk} = \mathsf{sk}_1 \oplus \cdots \oplus \mathsf{sk}_n$ hardwired in it. $C_{[\mathsf{hk}, y_i, \mathsf{sk}]}$ is defined in Figure 2.

5. Each party i performs the following steps: For every $j \in [L]$, compute $\pi_j = \mathsf{SSB.Open}(\mathsf{hk}, x_i, j)$ and the output bit $\mathsf{out}_j = \tilde{C}_i(j, x_i[j], \pi_j)$. Output $\mathsf{out} = \mathsf{out}_1, \ldots, \mathsf{out}_L$.

[9][41] give a construction of SSB hash from an FHE scheme.

Input: $j \in [L]$, $c \in \{0,1\}^{\ell_{\mathsf{ctx}}}$, $a \in \{0,1\}$, $\pi \in \{0,1\}^{\ell_{\mathsf{opn}}}$.
Constants: Hash key hk, hash value y_i, and decryption key sk.

(a) Check that $\mathsf{SSB.Verify}(\mathsf{hk}, y_i, j, (c,a), \pi) = \mathsf{accept}$. If not, output 0.

(b) Output $\mathsf{FHE.Dec}_{\mathsf{sk}}(c)$.

Figure 2: Function $C_{[\mathsf{hk},y_i,\mathsf{sk}]}\left(j, (c,a), \pi\right)$

This completes the description of $\Pi_{\text{long-out}}$. It is easy to verify that $\Pi_{\text{long-out}}$ satisfies our desideratum for communication efficiency. In particular, note that the use of standard MPC protocol Π in $\Pi_{\text{long-out}}$ is only for computing functionalities with "small" output that are independent of the function f.

Lemma 6.1. *If fully homomorphic encryption and indistinguishability obfuscation exist, then* $\Pi_{\text{long-out}}$ *is a secure MPC protocol against semi-honest adversaries.*

The proof of security follows by a straightforward extension of the proof in [41]. We omit the details from this manuscript.

Compressing Randomness of Parties in CRS Model. We now describe a simple, generic procedure to compress "long" local randomness into "short" local randomness, in the CRS model.

Let $\Pi_{\text{long-rand}}$ be any semi-honest MPC protocol for computing general n-party functions where the size of randomness of each party P_i grows with the function output length. Fix an n-party function f with (maximum) output length ℓ_{out}. Let $\ell_{\text{rand}} = \ell_{\text{rand}}(\ell_{\text{out}})$ be the size of randomness of each party in $\Pi_{\text{long-rand}}$ for computing f.

We transform $\Pi_{\text{long-rand}}$ into a new semi-honest protocol $\Pi_{\text{short-rand}}$ in the CRS model where the size of the randomness of each party P_i is independent of the function output length. Let PRG be a pseudorandom generator that stretches k bits into ℓ_{rand} bits. The length of randomness of each party in $\Pi_{\text{long-rand}}$ will be simply k; however, the length of the CRS will be $\ell_{\text{crs}} = n \cdot \ell_{\text{rand}}$ where n is the number of parties.

We now describe protocol $\Pi_{\text{short-rand}}$:

- Denote the CRS as $\mathsf{CRS} = \mathsf{CRS}_1, \ldots, \mathsf{CRS}_n$.
- Let r_i denote the randomness of party P_i. At the beginning of the protocol, party P_i locally computes $R_i = \mathsf{PRG}(r_i) \oplus \mathsf{CRS}_i$.
- For the remainder of the protocol, P_i follows the strategy of party i in $\Pi_{\text{long-rand}}$ using randomness R_i.

The correctness and efficiency properties of $\Pi_{\text{short-rand}}$ are easy to verify.

Lemma 6.2. *If* $\Pi_{\text{long-rand}}$ *is a semi-honest MPC protocol for computing an n-party function f and* PRG *is a secure pseudorandom generator, then* $\Pi_{\text{short-rand}}$ *is a semi-honest MPC protocol for f in the CRS model.*

Proof. (Sketch) Let $M \subset [n]$ denote the set of corrupted parties. Let $\mathcal{S}_{\text{long-rand}}$ be the simulator for $\Pi_{\text{long-rand}}$. Given, $\mathcal{S}_{\text{long-rand}}$, we build a simulator $\mathcal{S}_{\text{short-rand}}$ for $\Pi_{\text{short-rand}}$ in the following manner. $\mathcal{S}_{\text{short-rand}}$ first runs $\mathcal{S}_{\text{long-rand}}$ to obtain a simulated protocol transcript $\tilde{\mathsf{trans}}$ along with randomness $\tilde{R}_{i_1}, \ldots, \tilde{R}_{i_{|M|}}$ for the corrupted parties $i_1, \ldots, i_{|I|} \in I$. For every $j \in I$, $\mathcal{S}_{\text{short-rand}}$ now chooses $\tilde{r}_j$ and $\tilde{\mathsf{CRS}}_j$ such that $\mathsf{PRG}(\tilde{r}_j) \oplus \tilde{\mathsf{CRS}}_j = \tilde{R}_j$. For every $j \in [n] \setminus M$, $\mathcal{S}_{\text{short-rand}}$ chooses $\tilde{\mathsf{CRS}}_j$ as a random string. Finally, $\mathcal{S}_{\text{short-rand}}$ outputs $\left(\tilde{\mathsf{CRS}}, \tilde{\mathsf{trans}}, \{\tilde{r}_j\}_{j \in M}\right)$, where $\tilde{\mathsf{CRS}} = \tilde{\mathsf{CRS}}_1, \ldots, \tilde{\mathsf{CRS}}_n$. The correctness of simulation is easy to verify. □

We remark that our transformation only works in the semi-honest model. In the malicious model, the corrupted parties may choose their inputs and randomness adaptively based on the CRS. In this case, the simulation strategy described above does not work. We stress, however, that this is *not* an issue for us since semi-honest security actually suffices for our purposes.

Protocol $\Pi_{\text{out-ind}}$ in CRS Model. Applying the transformation in Lemma 6.2 on the MPC protocol from Lemma 6.1, we obtain a semi-honest protocol $\Pi_{\text{out-ind}}$ in the CRS model where the size of the randomness of the parties as well as the communication complexity of $\Pi_{\text{out-ind}}$ is independent of the function output length.

Lemma 6.3. *Assuming the existence of indistinguishability obfuscation and fully homomorphic encryption, protocol $\Pi_{\text{out-ind}}$ obtained by applying the transformation from Lemma 6.2 on protocol $\Pi_{\text{long-out}}$ from Lemma 6.1 is secure against semi-honest adversaries. The randomness of each party in $\Pi_{\text{out-ind}}$ is of fixed length k and the communication complexity of $\Pi_{\text{out-ind}}$ is* $\text{poly}(k, n) + n \cdot \ell_{\mathsf{inp}}(k)$, *where ℓ_{inp} denotes the input length of each party.*

Reusable Correlated Randomness in CRS Model. Let $\mathcal{CR}_{\text{nr}}$ denote any non-reusable correlated randomness setup for n parties. We now describe a procedure that transforms $\mathcal{CR}_{\text{nr}}$ into a new correlated randomness setup $\mathcal{CR}$ that can be used $L = \text{poly}(k)$ times for any a priori fixed polynomial L. This transformation works in the CRS model, i.e., in order to use the new correlated randomness setup, we require that the parties have access to a CRS where the size of the CRS is linear in L.

Let Samp denote the sampling algorithm that generates a single instance of $\mathcal{CR}_{\text{nr}}$. That is, Samp takes as input a random string z of length k and outputs a tuple $(r_1, \ldots, r_n)$, where r_i is the (correlated) randomness for party i. For simplicity of notation, we assume that each r_i is of the same length k. Let PRG be a pseudorandom generator that stretches a random seed of length k into $k \cdot L$ pseudorandom bits.

We first define a deterministic functionality F_{cr} that takes as input n random string pairs $(s_1, m_1, \ldots, s_n, m_n) \in \{0,1\}^k$ and outputs L independent instances of $\mathcal{CR}_{\text{nr}}$. See Figure 3 for a formal description of F_{cr}.

We now define $\mathcal{CR}$. To sample an instance $r_1, \ldots, r_n$ of $\mathcal{CR}_{\text{nr}}$, we first sample an *honest* execution of protocol $\Pi_{\text{out-ind}}$ for computing F_{cr} in the CRS model. Let view_i denote the view of party i in the execution where view_i consists of its private randomness (that includes the seed m_i) and the protocol messages. The (correlated) randomness of party i is simply set to $r_i = \mathsf{view}_i$.

Given this correlated randomness and access to the CRS used in the execution of $\Pi_{\text{out-ind}}$, party i first uses view_i and CRS to reconstruct the output of $\Pi_{\text{out-ind}}$, namely, $\left\{R_1^j\right\}_{j=1}^{L}, \ldots, \left\{R_n^j\right\}_{j=1}^{L}$. Next, it computes the masks $m_i^1, \ldots, m_i^L \leftarrow \mathsf{PRG}(m_i)$. Finally, it computes $\left\{r_i^j\right\}_{j=1}^{L}$ where $r_i^j =$

1. Compute $s = s_1 \oplus \cdots \oplus s_n$.
2. Compute $z_1, \ldots, z_L \leftarrow \mathsf{PRG}(s)$.
3. For every $j \in [L]$, compute a correlated randomness instance $(r_1^j, \ldots, r_n^j) \leftarrow \mathsf{Samp}(z_j)$.
4. For every $i \in [n]$, compute "masks" $m_i^1, \ldots, m_i^L \leftarrow \mathsf{PRG}(m_i)$.
5. For every $i \in [n]$, $j \in [L]$, compute $R_i^j = r_i^j \oplus m_i^j$.
6. Output $\left\{R_1^j\right\}_{j=1}^{L}, \ldots, \left\{R_n^j\right\}_{j=1}^{L}$.

Figure 3: Function $F_{\text{cr}}(s_1, m_1, \ldots, s_n, m_n)$

$\tilde{r}_i^j \oplus m_i^j$. Note that r_i^j denotes the jth instance of $\mathcal{CR}_{\text{nr}}$. Thus, from a single instance of $\mathcal{CR}$, we are able to generate L instances of $\mathcal{CR}_{\text{nr}}$.

We note that the use of masks ensures that party i can only reconstruct its own shares of the correlated randomness while the shares of the other parties remain hidden from its view.

Computing General Functions with Reusable Correlated Randomness in the CRS Model. Let $\Pi_{\text{gen}}^{\text{nr}}$ denote the protocol from Theorem 18 for computing arbitrary polynomial-time functions with general interaction patterns in the non-reusable correlated randomness setup model. Applying our transformation from above to the correlated randomness setup of $\Pi_{\text{gen}}^{\text{nr}}$, we obtain a new protocol Π_{gen} with a reusable correlated randomness setup in the CRS model.

Theorem 19. *Assuming indistinguishability obfuscation for general circuits and fully homomorphic, for every interaction pattern $\mathcal{I}$, there exists an $\mathcal{I}$-compliant protocol for computing any polynomial-time function that achieves malicious security against any number of corruptions. The protocol requires a reusable correlated randomness setup in the CRS model where the size of the CRS grows linearly with the number of uses of the protocol.*

Remark 1. *We remark that the long CRS in the above result can be easily compressed into a short CRS in the random oracle model, via standard techniques.*

7 Computational Protocols for Symmetric Functions

We describe simpler protocols for computing symmetric functions with semi-honest security, which do not not require obfuscation. These protocols use multilinear maps, and their security can be reduced to a simple DDH-like assumption for the underlying maps. We note that current multilinear-map candidates are not strong enough to support this application, indeed the protocols below are insecure using any of the known construction [30, 20, 33, 21]. Nonetheless we present this protocols as a natural application of multilinear maps, as it is plausible that future candidates will be able to support these protocols.

Notation. We review very briefly the notion of "multilinear maps" that we use, cf. [30]. Recall that a graded encoding scheme (in the symmetric setting, with public sampling) with n levels allows one to sample pairs $(a, [a]_1)$, with a a random plaintext element and $[a]_1$ an encoding of a at level 1. More generally we denote a level-i encoding of a by $[a]_i$.

Encodings can also be publicly negated, added and multiplied: negation does not change the level, addition of two same-level encoding yields an encoding of the sum at the same level, and multiplication of level-i and level-j encodings yield an encoding of the product at level $i + j$, provided that $i+j \leq n$. Encoding can also be multiplied by plaintext elements, which are considered level-0 encoding for this purpose. Finally, a scheme like that comes equipped with a zero-testing procedure, allowing anyone to test whether a level-n encoding encodes zero. In our context we also need the encoding scheme to be *level-n randomizable*, i.e. given an encoding $[a]_n$ it should be possible to generate a new random level-n encoding of the same element a. Below we denote addition, subtraction and multiplication, respectively, of encodings by overloading the operators $(+, -, \times)$.

7.1 Protocol for Star Pattern

Protocol Π_{star}. We describe a protocol Π_{star} that works with an n-level graded encoding scheme.

Every party i gets level-1 encoding of two elements $[a_i]_1$ and $[a_i \times r]_1$, all using the same ratio r. Denote the product of the a_i's by $A = \prod_{i=1}^{n} a_i$, the evaluator gets level-k encoding of all the elements $[A \times r^i]_n$ for $i = 0, 1, \ldots, n$ in random order, and with each $[A \times r^i]_n$ it gets the function value $f_i = f(1^i 0^{n-i})$.

Later, each party i with input b_i send to the server the encoding of $[a_i \times r^{b_i}]_1$ (namely $[a_i]_1$ if $b_i = 0$ and $[a_i \times r]_1$ if $b_i = 0$). The evaluator multiplies all these level-1 encodings, thus getting a level-n encoding $\left[\prod_i a_i \times r^{\sum_i b_i}\right]_n$. It then subtracts this element from all the elements on its list, uses zero-testing to find the index that it matches, and outputs the corresponding function value f_i.

This completes the description of Π_{star}. We reduce the security of the star protocol in the semi-honest, static corruption model, to the following MDDH-like assumption, which we call bookend-MDDH

The bookend-MDDH assumption. Consider a symmetric n-level graded encoding scheme that supports public sampling and re-randomization at levels 1 and n, we assume that for every $1 \leq s < n$, the following two distributions are indistinguishable:

$$\left(\mathsf{params}, [1]_1, [r]_1, ([a_i]_1 : i = 0, \ldots, s-1), ([A/r^i]_n : i \in [s]), [Ar^{n-s+i}]_n : i \in [s])\right),$$

$$\text{and } \left(\mathsf{params}, [1]_1, [r]_1, ([a_i]_1 : i = 0, \ldots, s-1), ([u_i]_n : i \in [s]), ([v_i]_n : i \in [s])\right)$$

with r and the a_i's, u_i's and v_i's chosen at random, and $A = \prod_{i=1}^{s} a_i$.

(The name bookend-MDDH comes because the scheme lets us compute all the elements $[ar^i]_n$ for $i = 0, \ldots, n-s$, but yet we assume that for the "bookends" $i = -s, \ldots, -1$ and $i = n-s+1, \ldots, n$ we cannot distinguish $[ar^i]_n$ from random.)

Theorem 20. *Assuming the bookend-MDDH assumption on multilinear maps, the proposed star-compliant protocol* Π_{star} *securely computes any symmetric function against semi-honest adversaries.*

Proof. Fix an n-input symmetric function $f(\cdot)$ and denote $f_i = f(1^i 0^{n-1})$ for $i = 0, 1, \ldots, n$. For ease of notations we assume (wlog) that the honest parties are the first n parties, and let x_0, x_1 be two honest-party inputs with the same residual function, $f(x_1|y) = f(x_2|y)$ for all y. We need to show that in this case the view of the adversary is indistinguishable between the cases where the honest parties' input is x_1 or x_2. Assume that we have an adversary $\mathcal{A}$ that can distinguish these two views, and we show how to use it to break the bookend-MDDH assumption from above.

The reduction algorithm $\mathcal{B}$ gets as input $\big(\mathsf{params}, [1]_1, [r]_1, ([a_i]_1 : i = 0, \ldots, s), ([u_i]_n : i = 0, \ldots, s), ([v_i]_n : i = 0, \ldots, s)\big)$, and it needs to decide if the u_i's and v_i's were chosen at random or set as $u_i = A/r^i$ and $v_i = Ar^{n-s+i}$ for $A = \prod_j a_j$.

$\mathcal{B}$ uses the sampling procedure of the scheme to sample a_i's also for the adversarial parties, getting random elements and their level-1 encoding, $\{(a_i, [a_i]_1) : i = s, \ldots, n-1\}$. It computes a level-1 encoding or $a_i \times r$ for $i = s, \ldots, n-1$ by multiplying $a_i \times [r]_1$ and then re-randomizing, and gives the pairs $([a_i]_1, [a_i \times r]_1)$ to $\mathcal{A}$ as the shares of the correlated state for the adversarial parties.

For the share of the evaluator, $\mathcal{B}$ choose at random $t \in_R \{1, 2\}$ and, and let w be the Hamming weight of the honest parties' input x_t. $\mathcal{B}$ computes level-n encoded values z_i, $i = 0, 1, \ldots, n-1$, using the u_i's for the first w of them, computing the next $n - s + 1$ as $r^i \times \prod_{i<s} a_i$, and using the v_i's for the last $n - w - s$. Specifically, $\mathcal{B}$ sets

$$z_i \leftarrow \begin{cases} u_{w-i} & \text{for } i \le w-1 \\ v_{i-n+s-w-1} & \text{for } i > n-s+w \\ ([r]_1)^{i-w} \times ([1]_1)^{n-s+w-i} \times \prod_{j<s}[a_j]_1 & \text{otherwise} \end{cases}$$

Finally, $\mathcal{B}$ multiply in the plaintext a_j's for $j \le s$ and re-randomize, setting $z_i' \leftarrow \mathsf{reRand}(z_i \times \prod_{j \ge s} a_i)$, and give $\mathcal{A}$ the pairs (z_i, f_i) in random order.

For the messages of the honest parties $i = 0, 1, \ldots, s-1$, $\mathcal{B}$ just uses the level-1 encodings $[a_i]_1$ from its input. After $\mathcal{A}$ receives all these inputs, it issues a guess as to whether the honest parties' input was x_1 or x_2. Then $\mathcal{B}$ outputs "real" if the guess matches its chosen index t and "random" otherwise.

`Analysis (sketch).` It can be seen that if the input of $\mathcal{B}$ was "real", i.e., $u_i = A/r^i$ and $v_i = A \times r^{n-s+i}$, then the view of the adversary is indeed consistent with a honest execution of the protocol with the honest parties' input being x_t (upto the statistical distance between re-randomized encodings and fresh ones).

On the other hand, if the input of $\mathcal{B}$ was "random" then the view of $\mathcal{A}$ is independent of the index t. This is because all the elements in the list of the evaluator that cannot be verified using the graded-scheme operations are random and independent in this case, regardless of what t is.

It therefore follows that the advantage of $\mathcal{B}$ in distinguishing real from random is exactly half the advantage of $\mathcal{A}$ in distinguishing x_1 from x_2. □

7.2 Protocol for General Interaction Patterns

Combining our protocol for star network with our general reduction in Section 3, for every interaction pattern $\mathcal{I}$, we obtain a semi-honest $\mathcal{I}$-compliant protocol Π_{gen} for computing symmetric functions with general interaction patterns.

Theorem 21. *Assuming the bookend MDDH assumption on multilinear maps, protocol Π_{gen} securely computes symmetric functions against semi-honest adversaries in the non-reusable correlated randomness setup model.*

8 Barriers for Efficient Information Theoretic Protocols

In this section, we present some negative results on the existence of communication-efficient protocols, even for very simple communication patterns such as two-chains entering an evaluator. Our negative results are conditional: we show that the existence of such efficient protocols would imply strong upper bounds, much better than what is known [18, 54, 27, 4], on the well-studied problem of (3-server, information-theoretic) Private Information Retrieval (PIR).

We start by considering a very simple network $\mathcal{S}_2$: the star network (see Section 4.2) with two parties P_0, P_1, each holding n-bit input (x_0, x_1, respectively), and an evaluator E who wish to compute $f(x_0, x_1)$, for some arbitrary function f. The following theorem claims that, even if we allow exponential randomness, a solution with polynomial communication complexity, for all f, implies 3-server PIR with polylogarithmic communication complexity.

Theorem 22. *Assume that, for every $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, there exists a semi-honest (statistically) secure protocol that computes f on the network $\mathcal{S}_2$, with randomness complexity $r(n)$ and communication complexity $c(n)$. Then, there exists an (interactive, statistical) 3-server PIR protocol, with communication complexity $O(c(\log N) + \log N + \log 1/\epsilon)$, where N is the database size and ϵ is the desired statistical security parameter for the PIR protocol.*

Proof. For clarity of presentation, we will start by making the stronger assumption that there is a *perfectly* secure protocol that computes any f on the network $\mathcal{S}_2$. Then, we will modify the proof to deal with the more involved case of statistical security.

As a first step, we argue that the assumption of the theorem implies that, for every f as above, there exists a secure protocol in the PSM model [28] with communication complexity $c(n)$ and *shared* randomness for the two clients of size $r'(n) = 2r(n)$ (and no randomness for the server). Specifically, given a protocol Π_f to compute f in the correlated-randomness model, on the network $\mathcal{S}_2$, that uses correlated randomness r_0, r_1, r_E (for P_0, P_1, E, respectively), we construct Π'_f by fixing r_E to some possible value (that has positive probability) and giving shared randomness (r_0, r_1), sampled from the conditional probability obtained by fixing r_E, to both P_0, P_1. The protocol Π'_f then proceeds as in Π_f. Security in the PSM model only requires that E does not obtain additional information beyond the output and, indeed, E's view in Π'_f is identical to its view in Π_f, when its randomness is r_E (here we use the perfect security of the protocol; we will get rid of this assumption later).

In our transformation of PSM protocols into PIR protocols, that will be presented below, the shared randomness will be picked by one of the servers and communicated. It is therefore important for us to reduce the length of the shared randomness used by the PSM protocol, even at the cost of slightly hurting the security. For this, we use a standard "sub-sampling" technique, where rather than picking $r'(n)$-bit randomness according to some generation process $\mathcal{G}$, a small multi-set $\mathcal{R} \subset \{0,1\}^{r'(n)}$ of random strings is fixed and one of them is uniformly chosen (using $\log |\mathcal{R}|$ bits) as the shared randomness. We say that $\mathcal{R}$ ϵ-fools a function $C : \{0,1\}^{r'(n)} \to [M]$ (wrt $\mathcal{G}$) if the statistical distance between $C(\mathcal{R})$ and $C(\mathcal{G})$ is bounded by ϵ, where $C(\mathcal{R})$ is the random variable obtained by applying C to a random element of $\mathcal{R}$ and $C(\mathcal{G})$ is the random variable obtained by applying C to a random $r'(n)$-bit string sampled by $\mathcal{G}$. We say that $\mathcal{R}$ ϵ-fools a family of functions $\mathcal{C}$ (wrt $\mathcal{G}$) if it ϵ-fools every function $C \in \mathcal{C}$. We use the following claim from [42, Lemma 1].[10]

[10]The original lemma is formulated for the case where $\mathcal{G}$ is uniform over $r(n)$-bit strings, but the proof remains the same, as each member of $\mathcal{R}$ is picked according to $\mathcal{G}$. The proof is a standard argument involving a Chernoff bound and a union bound.

Claim 8.1. *Let $\mathcal{C}$ be a family of functions from $\{0,1\}^{r'(n)}$ to $[M]$ and let $\epsilon > 0$. Then, there exists a set $\mathcal{R}_{\mathcal{C}} \subset \{0,1\}^{r'(n)}$ of size* $\mathrm{poly}(1/\epsilon, M, \log|\mathcal{C}|)$ *that ϵ-fools $\mathcal{C}$.*

If we let $M \leq 2^{c(n)}$ be the number of possible communications and, for each input (x_0, x_1), we have a function $C_{(x_0,x_1)}(r)$ that maps the input to the corresponding communication (depending on the randomness r), then $|\mathcal{C}| = 2^{2n}$ and $|\mathcal{R}_{\mathcal{C}}| = \mathrm{poly}(1/\epsilon, 2^{c(n)}, n)$. The modified protocol uses $\log|\mathcal{R}_{\mathcal{C}}|$ bits of randomness, its communication complexity is at most $c(n)$, and the view of E on any input is at most ϵ-far from its view on the same input in the original protocol. Denote the resulting protocol by Π''_f.

Finally, we obtain a 3-server PIR protocol for 3 servers $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$, each holding N-bit database D, and a user $\mathcal{U}$ that holds $i \in [N]$. The PIR protocol works as follows:[11]

1. Server $\mathcal{S}_1$ picks shared randomness $r \in \mathcal{R}$ (this depends on the function f being computed, that depends on the database D; see below) and sends r to user $\mathcal{U}$.

2. $\mathcal{U}$ additively shares i (i.e., $i = a + b \bmod N$) and also picks a random bit c. It sends (a, c) to $\mathcal{S}_1$ and (b, r) to $\mathcal{S}_2$.

3. Servers $\mathcal{S}_1, \mathcal{S}_2$ simulate the clients in a PSM protocol Π''_{f_D} to compute the function $f_D((a,c), b) = D[a + b \bmod N] \oplus c$ (using shared randomness r). They send their messages to server $\mathcal{S}_3$ who simulates the PSM referee (and who also knows f_D). $\mathcal{S}_3$ recovers the output bit and sends it to $\mathcal{U}$.

4. $\mathcal{U}$ unmasks $D[i]$ by xoring c.

The correctness is obvious. The communication complexity consists of $O(\log|\mathcal{R}|) = O(\log 1/\epsilon + c(\log N) + \log N)$ bits for distributing the randomness r, then $O(\log N)$ bits for sending a, b, and $c(\log N)$ bits for computing f_D (whose inputs are of length $\log N$). Altogether, as promised in the theorem statement. Finally, to argue security, servers $\mathcal{S}_1, \mathcal{S}_2$ just see sharing of i. Server $\mathcal{S}_3$ has exactly the same view as the referee in the PSM for computing f_D. The output of f_D reveals nothing to $\mathcal{S}_3$, due to the random mask c.

So far, we handled the case of perfect security. Next, we will deal with the case of statistical security by making the necessary modifications to the above proof. The first step would be, as above, to move from the correlated randomness setting to the PSM (shared randomness) setting. The difficulty here is how to pick r_E. While we know that the observer's views on any two inputs with the same output are δ-close, which implies that there exists randomness r_E for E where the conditioned views will also be δ-close, this randomness may be different for any pair of inputs and, moreover, it will be more convenient to get rid of the conditioning on the output value. In light of this, we will first restrict our attention to functions of the form $f'((x, c), y) = f(x, y) \oplus c$, where c is a bit that will be picked at random. Observe that this is the type of functions that we actually use in our PSM to PIR transformation. Moreover, this means that for every pair of inputs $(x, y), (x', y')$ (excluding the bit c) the views of E in the protocol for f' are δ-close (such a view consists of E's randomness, r_E, and the messages it receives from both clients, denoted as a random variable $M(x, y)$). This in turn means that, the distributions $((x, y), r_E, M(x, y))$ and $((x, y), r_E, M(x', y'))$ are δ-close, for any $(x, y), (x', y')$ and hence also if $(x, y), (x', y')$ themselves are picked uniformly at random. Therefore, there exists an r_E such that, conditioned on its choice, the distributions $((x, y), M(x, y))$ and $((x, y), M(x', y'))$ are δ-close, for random $(x, y), (x', y')$. We fix such r_E and,

[11] A similar transformation is given in [28] to obtain a 3-oracle, interactive, instance hiding scheme for all functions.

as before, we give shared randomness (r_0, r_1) (sampled from the conditional probability obtained by fixing r_E) to both P_0, P_1. The protocol $\Pi'_{f'}$ then proceeds as in $\Pi_{f'}$.

In the next stage, as before, we reduce the amount of shared randomness that the protocol uses (at the price of slightly hurting the security), and it proceeds similarly to the perfect case. We choose the same parameters as above (in particular, note that the function $C_{(x,y)}(r)$ is essentially the same as the random variable $M(x, y)$, viewed as a function of the randomness r). Hence, choosing the shared randomness from the smaller set $\mathcal{R}_{\mathcal{C}}$, changes the distribution of each $M(x, y)$ by at most ϵ and therefore also $((x, y), M(x, y))$ and $((x, y), M(x', y'))$ are $(\delta+2\epsilon)$-close, for random $(x, y), (x', y')$ and randomness chosen from $\mathcal{R}_{\mathcal{C}}$. As before, we denote the resulting protocol for f' by $\Pi''_{f'}$.

Finally, we turn this PSM protocol into a PIR. The main difference is that now we only have a guarantee that refers to random inputs to the PSM. We will therefore make sure our PIR invokes the PSM with random inputs. We modify the above construction as follows. The user starts by picking a random shift for the database $\Delta \in_R \{0, \ldots, N-1\}$. To retrieve the i-th bit of D, the user will retrieve the $i' = i + \Delta$ bit of $D' = D \gg \Delta$. Since i' is uniformly distributed, we will have the security. In more detail, the protocol proceeds as the previous PIR protocol where the index is i' and the database is D'. Since i' is random then so are its shares (a, b). Also, as mentioned, the function $f_{D'}$ is of the right form. The correctness of the PIR protocol is obvious. As for its security, each of the servers $\mathcal{S}_1, \mathcal{S}_2$ just sees random values: one share of i' and Δ and r which are independent of i. Server $\mathcal{S}_3$ sees Δ and the view of the referee in the PSM for computing $f_{D'}$ (as before, the output of $f_{D'}$ reveals nothing to $\mathcal{S}_3$, due to the mask c). For any two indices i, j, the corresponding i', j' are uniformly distributed and therefore the corresponding sharings $(a_i, b_i), (a_j, b_j)$ are also uniformly distributed. Using the security properties of the PSM, we have that the view of $\mathcal{S}_3$ in the first case, i.e. $(M(a_i, b_i), (a_i, b_i))$ is $(\delta + 2\epsilon)$-close to $(M(a_i, b_i), (a', b'))$ (where (a', b') are random) which, in turn, is $(\delta+2\epsilon)$-close to $(M(a_j, b_j), (a_j, b_j))$. All together, the views are at distance $O(\delta + \epsilon)$. (For convenience we can pick $\delta = \Theta(\epsilon)$.) □

The above theorem refers to a setting where the network is small (2 parties and an evaluator) but the inputs are long (n-bit strings). The following simple corollary shows that similar results hold in the case where the network is large ($2n$ parties and an evaluator) but the inputs are short (single bits). Specifically, we consider a simple network $\mathcal{N}_n$ that consists of two chains of n parties each $P_0, \ldots, P_{n-1}$ and $Q_0, \ldots, Q_{n-1}$ where each of P_{n-1}, Q_{n-1} is connected to an evaluator E.

Corollary 8.2. *Assume that, for every $f : \{0, 1\}^{2n} \to \{0, 1\}$, there exists a semi-honest (statistically) secure protocol that computes f on the network $\mathcal{N}_n$, with randomness complexity $r(n)$ and communication complexity $c(n)$. Then, there exists (interactive, statistical) 3-server PIR protocol, with communication complexity $O(c(\log N) + \log N + \log 1/\epsilon)$, where N is the database size and ϵ is the desired statistical security parameter for the PIR protocol.*

Proof. The proof is by a simple reduction to the previous case. If the assumption of the corollary holds, then also the assumption of Theorem 22 holds: to compute any function f on $\mathcal{S}_2$ the parties will simulate the computation of a corresponding function f' on the network $\mathcal{N}_n$, where the n-bit input of the first party in $\mathcal{S}_2$ is distributed among the n players $P_0, \ldots, P_{n-1}$ of the first chain and, similarly, the n-bit input of the second party in $\mathcal{S}_2$ is distributed among the n players $Q_0, \ldots, Q_{n-1}$ of the second chain. The correctness is obvious, the communication is only smaller, and the view of each set in $\mathcal{S}_2$ corresponds to the view of an appropriate set in $\mathcal{N}_n$. The corollary then follows from the theorem. □

References

[1] S. Agrawal, S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption: New perspectives and lower bounds. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 500–518, 2013.

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

[3] A. Beimel, A. Gabizon, Y. Ishai, E. Kushilevitz, S. Meldgaard, and A. Paskin-Cherniavsky. Non-interactive secure multiparty computation. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 387–404, 2014.

[4] A. Beimel, Y. Ishai, E. Kushilevitz, and I. Orlov. Share conversion and private information retrieval. In *Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012*, pages 258–268, 2012.

[5] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796, 2012.

[6] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[7] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, pages 169–188, 2011.

[8] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography*, pages 253–273, 2011.

[9] D. Boneh and A. Silverberg. Applications of multilinear forms to cryptography. *IACR Cryptology ePrint Archive*, 2002:80, 2002.

[10] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[11] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145, 2001.

[12] R. Canetti, A. Cohen, and Y. Lindell. A simpler variant of universally composable security for standard multiparty computation. Cryptology ePrint Archive, Report 2014/553, 2014. http://eprint.iacr.org/.

[13] A. D. Caro, V. Iovino, A. Jain, A. O'Neill, O. Paneth, and G. Persiano. On the achievability of simulation-based security for functional encryption. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 519–535, 2013.

[14] N. Chandran, J. A. Garay, and R. Ostrovsky. Improved fault tolerance and secure computation on sparse networks. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, pages 249–260, 2010.

[15] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.

[16] J. H. Cheon, K. Han, C. Lee, H. Ryu, and D. Stehlé. Cryptanalysis of the multilinear map over the integers. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2015. http://eprint.iacr.org/2014/906.

[17] J. H. Cheon, C. Lee, and H. Ryu. Cryptanalysis of the new clt multilinear maps. Cryptology ePrint Archive, Report 2015/934, 2015. http://eprint.iacr.org/2015/934.

[18] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[19] J. Coron, C. Gentry, S. Halevi, T. Lepoint, H. K. Maji, E. Miles, M. Raykova, A. Sahai, and M. Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology - CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 247–266. Springer, 2015. Long version in http://eprint.iacr.org/2015/596.

[20] J. Coron, T. Lepoint, and M. Tibouchi. Practical multilinear maps over the integers. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 476–493. Springer, 2013.

[21] J. Coron, T. Lepoint, and M. Tibouchi. New multilinear maps over the integers. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 267–286. Springer, 2015.

[22] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.

[23] I. Damgård and S. Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *TCC*, pages 621–641, 2013.

[24] Y. Dodis, J. Katz, L. Reyzin, and A. Smith. Robust fuzzy extractors and authenticated key agreement from close secrets. In C. Dwork, editor, *Advances in Cryptology - CRYPTO'06*, volume 4117 of *Lecture Notes in Computer Science*, pages 232–250. Springer, 2006.

[25] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *J. ACM*, 40(1):17–47, 1993.

[26] C. Dwork, D. Peleg, N. Pippenger, and E. Upfal. Fault tolerance in networks of bounded degree. *SIAM J. Comput.*, 17(5):975–988, 1988.

[27] K. Efremenko. 3-query locally decodable codes of subexponential length. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 39–44, 2009.

[28] U. Feige, J. Killian, and M. Naor. A minimal model for secure computation (extended abstract). In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 554–563, New York, NY, USA, 1994. ACM.

[29] J. A. Garay and R. Ostrovsky. Almost-everywhere secure computation. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 307–323, 2008.

[30] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 1–17, 2013.

[31] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49, 2013.

[32] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.

[33] C. Gentry, S. Gorbunov, and S. Halevi. Graph-induced multilinear maps from lattices. In Y. Dodis and J. B. Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 498–527. Springer, 2015.

[34] O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications.* Cambridge University Press, 2004.

[35] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[36] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F. Liu, A. Sahai, E. Shi, and H. Zhou. Multi-input functional encryption. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 578–602, 2014.

[37] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 555–564, 2013.

[38] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[39] S. D. Gordon, T. Malkin, M. Rosulek, and H. Wee. Multi-party computation of polynomials and branching programs without simultaneous interaction. In *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 575–591. Springer, 2013.

[40] S. Halevi, Y. Lindell, and B. Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 132–150. Springer, 2011.

[41] P. Hubacek and D. Wichs. On the communication complexity of secure function evaluation with long output. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 163–172, 2015.

[42] Y. Ishai and E. Kushilevitz. On the hardness of information-theoretic multiparty computation. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 439–455, 2004.

[43] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer – efficiently. In *CRYPTO*, pages 572–591, 2008.

[44] M. Kearns, J. Tan, and J. Wortman. Network-faithful secure computation.

[45] J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[46] B. Minaud and P.-A. Fouque. Cryptanalysis of the new multilinear map over the integers. Cryptology ePrint Archive, Report 2015/941, 2015. http://eprint.iacr.org/2015/941.

[47] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 590–599, 2001.

[48] A. O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.

[49] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 457–473, 2005.

[50] N. B. Shah, K. V. Rashmi, and K. Ramchandran. Secret share dissemination across a network. *CoRR*, abs/1207.0120, 2012.

[51] A. Srinivasan and D. Zuckerman. Computing with very weak random sources. *SIAM J. Comput.*, 28(4):1433–1459, 1999.

[52] A. Yao. Protocols for secure computations. In Proceedings of FOCS'82, pages 160–164, Chicago, 1982. IEEE.

[53] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

[54] S. Yekhanin. Towards 3-query locally decodable codes of subexponential length. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 266–274, New York, NY, USA, 2007. ACM.