

# Towards Topology-Hiding Computation from Oblivious Transfer

Marshall Ball\*    Alexander Bienstock†    Lisa Kohl‡    Pierre Meyer§

June 6, 2023

## Abstract

*Topology-Hiding Computation (THC)* enables parties to securely compute a function on an incomplete network without revealing the network topology. It is known that secure computation on a *complete* network can be based on oblivious transfer (OT), even if a majority of the participating parties are corrupt. In contrast, THC in the dishonest majority setting is only known from assumptions that imply (additively) homomorphic encryption, such as Quadratic Residuosity, Decisional Diffie-Hellman, or Learning With Errors.

In this work we move towards closing the gap between MPC and THC by presenting a protocol for THC on general graphs secure against all-but-one semi-honest corruptions from *constant-round constant-overhead secure two-party computation*. Our protocol is therefore the first to achieve THC on arbitrary networks without relying on assumptions with rich algebraic structure. As a technical tool, we introduce the notion of *locally simulatable MPC*, which we believe to be of independent interest.

## 1 Introduction

A secure multi-party computation (MPC) protocol enables a set of mutually distrusting parties with private inputs to jointly perform a computation over their inputs such that no adversarial coalition can learn anything beyond the output of the computation. Results in the 1980s showed that, under widely-believed assumptions, any function that can be feasibly computed can be computed securely [Yao82, GMW87, BGW88, CCD88].

However, these early protocols and most of the subsequent work (as well as their corresponding security definitions), assume that the communication graph is a complete network: any two parties can communicate directly. In many situations communication networks are incomplete and, additionally, the structure of the communication network itself may be sensitive information which the participants desire to keep private (e.g. network topology may reveal information about users' locations, or relationships between users).

**Topology hiding computation.** Moran et al. [MOR15] noticed that there are situations where the communication network should additionally be kept private: secure computation over a social

---

\*New York University, USA. E-mail: [marshall.ball@cs.nyu.edu](mailto:marshall.ball@cs.nyu.edu).

†New York University, USA. E-mail: [abienstock@cs.nyu.edu](mailto:abienstock@cs.nyu.edu).

‡CWI, Cryptology Group, Amsterdam, The Netherlands. E-mail: [lisa.kohl@cwi.nl](mailto:lisa.kohl@cwi.nl).

§FACT Center, Reichman University, Israel and IRIF, Université Paris Cité, CNRS, France. E-mail: [pierre.meyer@irif.fr](mailto:pierre.meyer@irif.fr).

network about the social network, securely computing a function individual location data, low locality MPC [BBC+19]. Motivated as such, Moran et al. [MOR15] then formalized the notion of *topology-hiding computation (THC)*, where parties can securely compute a function without revealing anything about the communication network (graph), beyond the immediate neighbors they are communicating with and what can be derived from the output of the function computed (which might be either topology independent, such as a message broadcast, or topology dependent, such as a routing table). In general, we say that a protocol is topology-hiding with respect to a class of graphs, if nothing is revealed beyond membership in that parties only see their immediate neighborhood and wish to jointly compute a function without revealing anything about the graph topology beyond what can be derived from the output (which might be topology independent, e.g., a message broadcast, or topology dependent, e.g., a routing table).

It turns out that even simply *broadcasting* a message to all parties in topology-hiding manner (with no privacy guarantees on the information sent) is challenging, even in the semi-honest setting where adversarial parties are assumed to follow the protocol execution.<sup>1</sup> But exactly how difficult it is to construct THC protocols remains poorly understood. In this vein, a line of work has sought to investigate the following question:

*Is semi-honest MPC equivalent to semi-honest THC?  
Are additional assumptions required to make a secure computation topology-hiding?*

The feasibility of semi-honest MPC (for arbitrary functions) obeys a dichotomy based on the number of corruptions and following this we can collect the work on semi-honest THC into two categories.

- **Honest majority ( $< n/2$  corruptions):** In this regime, we know that semi-honest MPC (on fully connected networks) can be achieved information-theoretically [BGW88, CCD88, RB89]. For THC (on arbitrary, connected communication graphs) it has been shown that key agreement is necessary with even just *one* corruption [BBC+20]. On the other hand, information-theoretic THC with a single corruption is possible if (and only if) one is promised that the communication graph is two-connected [BBC+20] (albeit at high cost).  
For *single corruption*, key agreement is not just necessary but sufficient to achieve THC (on arbitrary connected graphs). [BBC+20] For a *constant* number of corruptions, THC is possible (on arbitrary connected graphs) assuming constant round MPC with constant computational overhead. [MOR15, BBMM18]
- **Dishonest majority ( $\geq n/2$  corruptions):** In the dishonest majority setting, no separation between MPC and THC is known. On the other hand, constructions of dishonest majority THC from general MPC (with a dishonest majority) are only known for very restricted graph classes [MOR15, BBMM18]: graphs of *constant diameter*.

Assuming *constant round MPC with constant computational overhead*,<sup>2</sup> THC is possible for graphs of *constant* degree and *logarithmic* diameter [MOR15, BBMM18].<sup>3</sup>

---

<sup>1</sup>In contrast, this is trivial to achieve (in the semi-honest setting) if hiding network topology is not a concern: simply forward the message through the network.

<sup>2</sup>MPC with constant computational overhead means that a circuit of size  $s(n)$  can be securely evaluated in time  $O(s(n)) + \text{poly}(\lambda)$ , where the latter term is a fixed polynomial of the security parameter.

<sup>3</sup>[HMTZ16] gave an early construction of a more efficient protocol for such graphs from the decisional Diffie-Hellman assumption.

THC for *arbitrary (connected) graphs* is only known from *structured hardness assumptions* (such as quadratic residuosity (QR), decisional Diffie-Hellman (DDH) and Learning with Errors (LWE)) [AM17, ALM17, LZM<sup>+</sup>18], or idealized obfuscation [BBMM18].

So while there is a clear separation between MPC and THC (with respect to general graphs) in the honest majority setting, no such separation is known in the dishonest majority setting. While OT is necessary and sufficient for MPC, it is unclear if it suffices to construct THC.<sup>4</sup> The motivation of this work is, thus, the following question:

*Are THC and MPC equivalent in the dishonest majority setting?*

## 1.1 Our Result

In this work, we make a step towards answering this question in the affirmative, by proving the following theorem:

**(Informal) Main Theorem 1** (Topology-Hiding Computation on All Graphs). *If there exists a two-party MPC protocol with constant rounds and constant computational overhead, then there exists a protocol securely realizing topology-hiding computation on every network topology in the presence of a semi-honest adversary corrupting any number of parties.*

The main feature of this construction is that it is the first construction of semi-honest topology-hiding computation tolerating any number of corruptions *on all graphs* from unstructured assumptions. As mentioned above, prior to this work it was only known how to construct THC against a semi-honest majority from constant round, constant computational overhead MPC for graphs with at most logarithmic diameter [MOR15, BBMM18], or from structured hardness assumptions [AM17, ALM17, LZM<sup>+</sup>18]. For the case of topology-hiding for general graphs, it was only known how to construct THC from constant round, constant computational MPC if the adversary was restricted to a *constant* number of corruptions [MOR15, BBMM18].

As an aside, our protocol is secure in the “pseudonymous neighbors” model (i.e. “knowledge-till-radius-zero”  $\text{KT}_0$  [AGPV88]), where parties only know pseudonyms of their neighbors (in this model, two colluding parties cannot determine if they share an honest neighbor). In contrast, Moran et al.’s protocol [MOR15] is only secure in the  $\text{KT}_1$  model (“knowledge-till-radius-one” [AGPV88]) where parties know globally consistent names for their neighbors (in this model, colluding parties can identify exactly which neighbors they have in common).

**On instantiating constant-round constant-overhead secure computation.** By [IKOS08], constant-round and constant-overhead two-party secure computation is implied by any constant-round OT protocol (which can be based, e.g., on the learning parity with noise (LPN) assumption [DDN14, YZ16, DGH<sup>+</sup>20], or on the computational Diffie-Hellman (CDH) assumption [BM90, DGH<sup>+</sup>20]) together with a constant-locality PRG with polynomial stretch (which can be based on a variant of an assumption by Goldreich [Gol00, MST03, OW14]).

---

<sup>4</sup>On the other hand, it is known that oblivious transfer is necessary to simply *communicate* in a topology-hiding manner in the presence of a dishonest majority. In particular, OT is implied by topology-hiding *broadcast* with a dishonest majority for graphs with just 4 nodes [BBMM18]. Again, because the broadcast functionality does not hide its inputs it is trivial to realize without hiding the topology. [BBC<sup>+</sup>20] showed that OT is necessary for topology-hiding *anonymous broadcast* on even simpler graphs.

In contrast, all previous constructions of THC for all graphs rely on structured hardness assumptions such as key-homomorphic encryption (“privately-key commutative and re-randomizable encryption, PKCR” [AM17, ALM17, LZM<sup>+</sup>18]), which does not seem to be implied by LPN/CDH and constant-locality PRGs (in fact, such a result would be rather surprising). We would like to point out though that the main focus of this work is not to build THC from different concrete assumptions, but to move away from structured assumptions, which are not necessary for secure computation without topology hiding, and—as we show in this work—are also not necessary for achieving topology-hiding computation on general graphs.

## 1.2 Organization of the paper

We provide an overview of our techniques and construction in Section 2. In Section 3, we introduce some preliminaries, including the definitions of THC in both the  $KT_0$  and  $KT_1$  models (the differences boils down to whether two corrupt parties know whether they share a common, honest, neighbor or not), as well as the constant-round, constant-overhead 2PC definition that we require. Then, in Section 4, we introduce our notion of *locally simulatable MPC*, and prove that protocols satisfying it are also *execution-oblivious*. Next, we introduce our protocol on directed paths and prove its local-simulatability in Section 5. Finally, we provide our full THB protocol on all graphs and show that it is indeed topology-hiding in Section 6.

## 2 Technical Overview

We first present a high-level overview of our techniques in Section 2.1, then present a more technical description of our core protocol in Section 2.2.

First, note that the difficulty in constructing protocols for THC can be reduced to the ability to perform topology-hiding broadcast (THB) of a single-bit message. Indeed, once parties can broadcast messages to the network in a topology-hiding way, one can use generic techniques that allow to establish secure computation given any OT protocol (leaking only the total number of nodes in the network). In the following overview, we therefore restrict ourselves to explaining how to achieve THB. With this simplification, we can capture our main result in the following theorem:

**Main Theorem (Informal)** (Topology-Hiding Computation on All Graphs). *If there exists a two-party MPC protocol with constant rounds and constant computational overhead, then there exists a topology-hiding protocol securely realizing broadcast on the class of all graphs in the presence of a semi-honest adversary corrupting any number of parties.*

For simplicity, we do not explicitly address the subtleties of the neighborhood models ( $KT_0$ , where neighbors are pseudonymized, or  $KT_1$ , where neighbor are identified “in the clear”) in this exposition, but the following high-level overview applies to both models.

### 2.1 A High-Level Overview

Our contribution is three-fold. First, we observe that many topology-hiding computation protocols implicitly follow the following informal paradigm: the parties run in parallel many instances of some (non topology-hiding on its own) subroutine, each one computing the desired function. Topology-hiding properties of the overall protocol emerge from the fact that the parties participate in each instance *obliviously*, meaning that each party is able to perform their role in each subroutine

without being able to identify which execution is which (even while colluding with other parties). Of particular interest is the protocol of Akavia et al. [ALM17, ALM20], which can be abstracted out as having the parties locally setup a mesh of *correlated random walks* along the topology, then perform some special-purpose MPC subprotocol along each path. In [ALM17, ALM20], these subroutines are instantiated by heavily leaning on assumptions with a rich algebraic structure. The first step in removing the need for these assumptions is to identify the properties we need from these MPC subroutines (or at least some sufficient properties we can instantiate from a form of oblivious transfer).

We then put forward the notion of *local simulation* as a sufficient security property to impose on these subroutines in order to allow for oblivious participant evaluation. A secure computation protocol over an incomplete network is *locally simulatable* if the view of each connected component in the adversary’s subgraph can be generated independently. As an example, in the network  $\textcircled{1}-\textcircled{2}-\textcircled{3}-\textcircled{4}-\textcircled{5}-\textcircled{6}-\textcircled{7}-\textcircled{8}$  (where parties  $\textcircled{2}$ ,  $\textcircled{3}$ ,  $\textcircled{6}$ , and  $\textcircled{7}$  are corrupt), the views of parties  $\{\textcircled{2}, \textcircled{3}\}$  and  $\{\textcircled{6}, \textcircled{7}\}$  should be simulated independently. Intuitively, this means the adversary cannot tell if  $\{\textcircled{2}, \textcircled{3}\}$  and  $\{\textcircled{6}, \textcircled{7}\}$  are participating in the same protocol or, *e.g.* in two different executions  $\textcircled{1}-\textcircled{2}-\textcircled{3}-\textcircled{4}-\textcircled{A}-\textcircled{B}-\textcircled{C}-\textcircled{D}$  and  $\textcircled{E}-\textcircled{F}-\textcircled{G}-\textcircled{H}-\textcircled{5}-\textcircled{6}-\textcircled{7}-\textcircled{8}$ . Ultimately, if using *correlated random walks*, that means that each party can participate in the MPC along each path without the adversary learning which chunk of walk corresponds to which other.

Finally, we provide a protocol for locally simulatable MPC on a path, assuming (semi-honest, static) secure two-party computation with *constant rounds* and *constant overhead*. By plugging this into the correlated random walks (*i.e.* the parties are obviously participating in a locally simulatable secure computation along each random walk), we obtain (dishonest majority, semi-honest, static) topology-hiding computation on all graphs. Previously, topology-hiding computation under this assumption was limited to the class of logarithmic-diameter graphs or to a constant number of corruptions on all graphs [MOR15].

We now expand (still at a high level) on each of these three points, without assuming familiarity with topology-hiding computation.

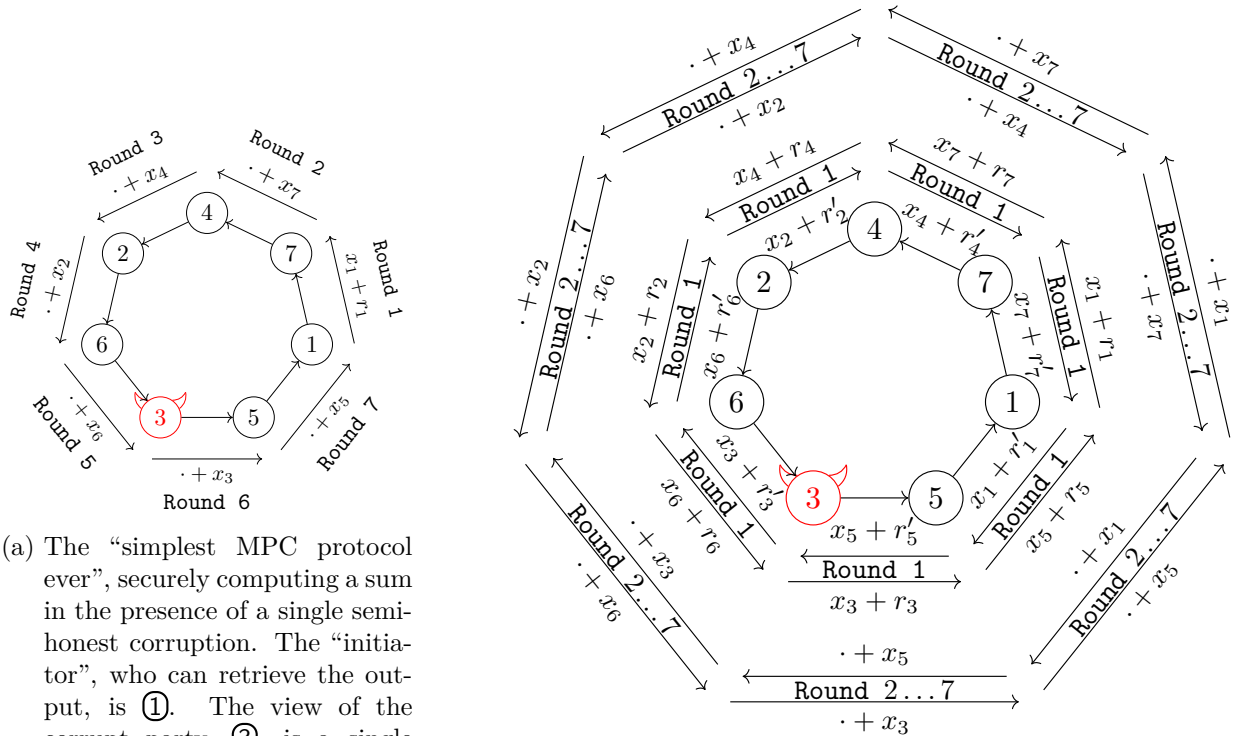
### 2.1.1 A modular approach to topology-hiding computation

Topology-hiding computation allows parties communicating through an incomplete network of point-to-point channels, where each party initially only knows their local neighborhood (possibly pseudonymized), to perform some secure computation without revealing any information about the network (beyond what they already know, *e.g.* each party’s respective neighborhood).

Our starting point is the observation that many topology-hiding protocols can be described informally in a very modular fashion, and yet their formal description (and the corresponding security proof) are inaccessibly monolithic. We start by a gentle introduction to this concept, with a modular presentation of the “simplest THC protocol”, realizing an information-theoretic topology-hiding sum in the presence of a single semi-honest corruption on cycles (more precisely, we fix a party/vertex set and consider all cycles on this set)<sup>5</sup>. Every party already knows they are on a cycle, but the secret part of the topology is the *order* in which they are arranged. We then provide

---

<sup>5</sup>In fact, the protocol we describe can be seen as a conceptually simpler alternative to Ball et al.’s [BBC<sup>+</sup>19, Theorem 4.1] 1-secure, semi-honest, information-theoretic topology-hiding anonymous broadcast on the class of all cycles with a given vertex set.



(a) The “simplest MPC protocol ever”, securely computing a sum in the presence of a single semi-honest corruption. The “initiator”, who can retrieve the output, is ①. The view of the corrupt party, ③, is a single message, obtained in round 5, masked by  $r_1$ .

(b) The “simplest THC protocol ever”, securely computing a sum in the presence of a single semi-honest corruption. In each round, the corrupt party ③’s view is comprised of two messages, masked by fresh one-time pads:  $r_6$  and  $r'_5$ , then  $r_2$  and  $r'_1$ , then  $r_4$  and  $r'_4$ , then  $r_7$  and  $r'_4$ , then  $r_1$  and  $r'_2$ , and finally  $r_5$  and  $r'_6$ .

Figure 1: The topology-hiding protocol of Figure 1b can be seen as running to  $2n$  parallel instances of the (non topology-hiding protocol) of Figure 1a.

a modular description of Akavia et al.’s [ALM17, ALM20] protocol, realizing (computational, semi-honest, dishonest majority) topology-hiding computation on the class of all graphs. The latter introduces the notion of *correlated random walks*, which form the basis for essentially all topology-hiding computation protocols *on all graphs*, tolerating any number of corruptions [ALM17, ALM20, LZM<sup>+</sup>18, Li22] (and now, also ours).

**An introductory example to modular THC.** Assume  $n$  parties are arranged in a cycle, each party only having access to a secure point-to-point channel with its neighbors in the cycle. Consider the following protocol (illustrated in Figure 1a), which is arguably the simplest (non topology-hiding) MPC protocol for securely computing a sum in the presence of a single semi-honest corruption. In the first round, an agreed upon party, which we will refer to as the *initiator*, samples a random value and uses it as a one-time pad to mask its input, then sends the resulting ciphertext to one of its neighbors (chosen arbitrarily). In each subsequent round, if a party received a message from one of its neighbors, it sums this message with its own input and passes on the result



to its other neighbor. After  $n$  rounds, the initiator receives the sum of all inputs masked by the one-time pad they themselves sampled, and they can therefore recover the desired output. Keeping in mind the parties are semi-honest and non-colluding, correctness and security are straightforward to verify (in essence, a single message is being passed around the cycle, containing the partial sum of previously visited parties’ inputs and masked by the initiators’ one-time pad). This only allows the initiator to get the output, however this can be addressed by running this “single-initiator” protocol  $n$  times sequentially with a fresh initiator for each instance.

As described, the protocol is not topology-hiding as, by noting in which round they receive a message, every party can learn their distance to the initiator, which leaks information about the graph. This can be addressed by considering the following augmented protocol (illustrated in Figure 1b):

- In the first round, each party samples two random masks, uses them as one-time pads for their input, and sends one of the resulting ciphertexts to each of its neighbors;
- In each subsequent round, every party receives two messages, one from each neighbor. Each party can add their input to these two messages, before forwarding them along the cycle (the message received from one neighbor is sent to the other neighbor, after the input is added).
- After  $n$  rounds, each party can receive the sum of all inputs by removing the appropriate mask from either of the messages received in the last round.

The above protocol could be described as running  $2n$  parallel instances of the “single-initiator” protocol. Each party’s instructions in the augmented protocol can be seen as participating in each of these protocols: in two of them with the role of initiator, in another two with the role of the party one hop away from the initiator, and so on for each of the  $n$  possible roles. Crucially, both for correctness and security, each party is able to participate in each subroutine *obliviously*, meaning that they are able to fulfill their role without being able to distinguish these executions and thus, most importantly, recognize which one corresponds to which initiator.

**Takeway for our protocol:** Skipping ahead, our main protocol will follow this abstract template: the parties will be participating in a slew of subroutines, where each party knows exactly their role in the process, but non-neighbor colluding parties cannot determine if they both participate in any given subroutine or not.

**Correlated random walks** [ALM17, ALM20]. In retrospect, the above protocol remains relatively simple to analyze, even without breaking it down into these subroutines. We now turn our attention to Akavia et al.’s [ALM17, ALM20] construction, for which taking a modular approach is significantly more interesting. In order to isolate Akavia et al.’s [ALM17, ALM20] key contribution of *correlated random walks*, we propose the following abstraction. Say there are  $n$  parties in some incomplete communication network wishing to securely compute an OR of their inputs. As a starting model, assume one party possesses an idealized hardware “black box”. This box is unclonable and has the following properties: any party may enter an input into the box, and after  $T$  inputs have been registered, the box returns their OR, where  $T$  is some parameter to be defined. The first party can place their input in the box, then pass on the latter to a randomly chosen neighbor. In subsequent rounds, the party who just received the box adds its input then passes the box to a randomly chosen neighbor. From a global point of view, the box is performing a random walk and

therefore, by known results on the *cover time* of a *simple random walk* in a connected graph, after  $T = \lambda \cdot n^3$  steps the box will have, with all but negligible probability, visited every party at least once each. This means that the last party will receive the correct OR from the box, and because we assumed the box was unclonable, the protocol securely computes OR<sup>6</sup>. This allows a randomly chosen<sup>7</sup> party to learn the output, and we could for instance sequentially repeat the process until every party obtains the OR. This protocol is not topology-hiding however, since colluding parties could learn an upper bound on their distance in the graph by counting the number of steps between when they handled the box<sup>8</sup>.

Akavia et al.’s [ALM17, ALM20] elegant solution is to have each party initially send a box to each of their neighbors. In every subsequent round, each party takes all the boxes it just received (one per neighbor), plugs in their input, then shuffles all boxes and sends one to each neighbor. After  $T$  rounds, each party opens any of the boxes it holds to recover the result. Observe that each box, taken individually, performs a random walk through the graph. While the walks of each box are not independent, but *correlated*, [ALM20, Lemma 3.14] establishes that setting  $T = \Theta(\lambda \cdot n^3)$  guarantees that with all but negligible probability all boxes will have individually covered the graph.

For completeness, we mention that in reality, Akavia et al. [ALM17, ALM20] do not rely on this idealized hardware to perform the secure OR on the fly, but use *linearly homomorphic Privately Key-Commutative and Rerandomizable encryption* (lhPKCR) [AM17], which can be instantiated from DDH [AM17], LWE [LZM<sup>+</sup>18], or QR [Li22]. In a nutshell, the parties pass around ciphertexts containing the homomorphically computed partial OR of the inputs of all visited nodes. In order to not have these ciphertexts be opened prematurely (*c.f.* the unclonability assumption on the black boxes), the secret key is re-randomized (and therefore secret-shared) along the walk: whenever a party receives a ciphertext they also “add a layer of randomization” to the key, which is possible for PKCR encryption. After  $\Theta(\lambda \cdot n^3)$  steps, when the random walk of every message is guaranteed to have visited every node with all but negligible probability, the parties return the ciphertexts to their source, along the reverse walks, and peeling off layers of encryption as they go.

**Takeway for our protocol:** Abstracting out, Akavia et al.’s [ALM17, ALM20] protocol can be seen as first having each party sample  $T$  permutations on their neighborhood (as illustrated in Figure 2, this globally define a mesh of random walks, where each party knows only their position), and then having the parties run a special-purpose MPC along each walk. These instances of MPC along each path are indistinguishable to the parties by using structural properties of lhPKCR encryption.

### 2.1.2 Information-local simulation.

Correlated random walks can be used to reduce the task of *topology-hiding computation on all graphs* to that of designing an MPC the parties can run along each walk without being to tell

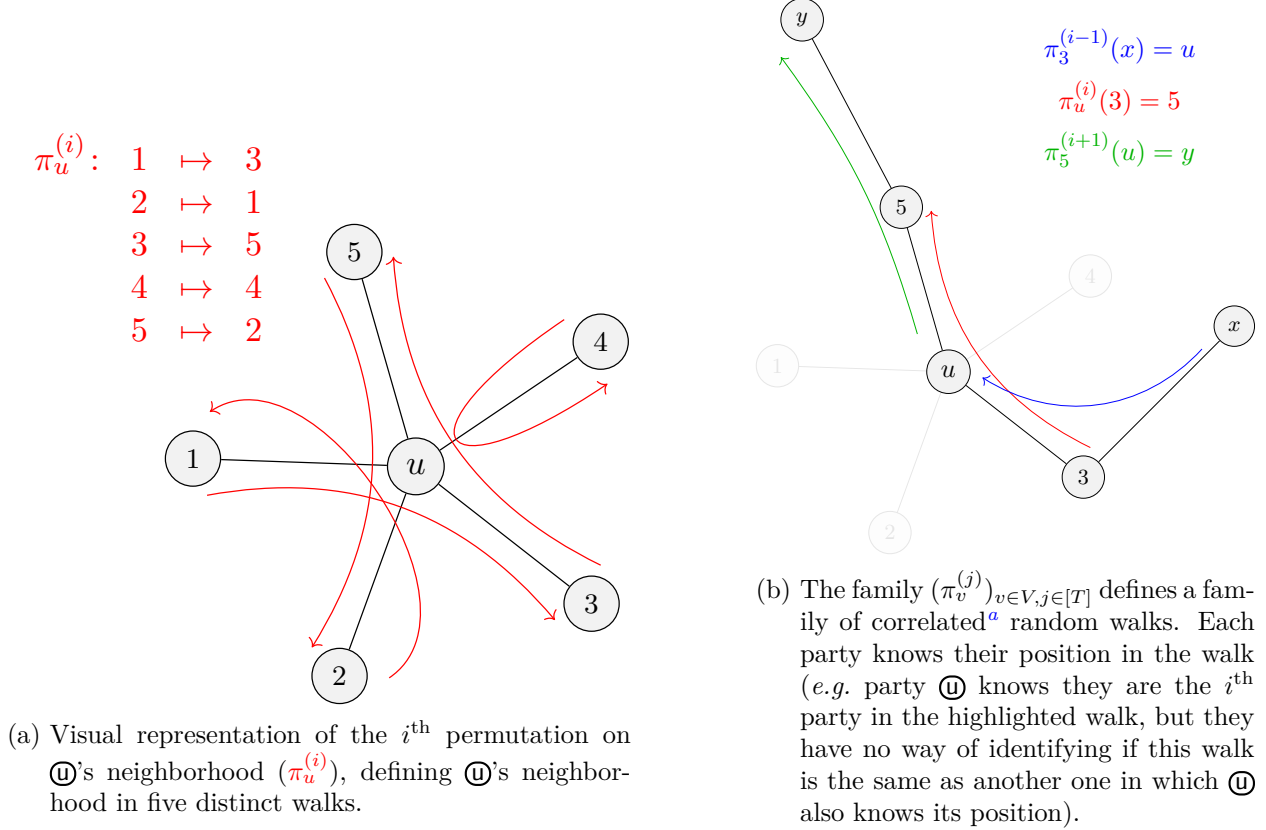
---

<sup>6</sup>If the box was clonable, a party could make a local copy then learn the partial OR of the inputs of all parties who previously handled the box by simply plugging 0s until they receive an output.

<sup>7</sup>The stationary distribution is not uniform, but nevertheless each party has non-negligible probability of being the last party.

<sup>8</sup>While this is beyond the scope of this exposition, we could quantify the leakage in terms of the *electric conductance* of the graph. What this means additionally is that the protocol is even insecure against a single corruption as a party can learn information from just counting the number of rounds between two consecutive visits of the box.





<sup>a</sup>The walks are *correlated* in the sense that no two walks borrow the same edge at the same time.

Figure 2: Local and global views of correlated random walks, obtained by having each party sampled uniformly at random  $T$  permutations on their neighborhood.

when they are participating in the same execution (i.e. in the same walk) or not. To instantiate the latter, we put forward the notion of *locally simulatable computation*.

**Introducing locally simulatable computation.** *Locally simulatable computation* is an MPC over an incomplete network  $G$  where the view of disconnected corrupt parties can be simulated *independently*. More formally consider the connected components  $\mathcal{Z}_1, \mathcal{Z}_2, \dots$  of the subgraph  $G[\mathcal{Z}]$  induced by the set of corrupted parties  $\mathcal{Z}$ . The views of the parties in each component  $\mathcal{Z}_i$  should be simulated given only their inputs, outputs, and local views of the graph, independently of the views of the parties in  $\mathcal{Z} \setminus \mathcal{Z}_i$ . Note that this requirement is orthogonal to the notion of being topology-hiding<sup>9</sup>:

- *THC is not necessarily locally simulatable:* Without loss of generality, a topology-hiding computation protocol can be made to be *not* locally simulatable, by first broadcasting a long

<sup>9</sup>However our final protocol will turn out to be both topology-hiding and locally simulatable.

random string (in a topology-hiding manner). The views of disconnected adversaries cannot be simulated independently, as they expect to receive the same string (which is not passed as input to the simulators, since it is neither an input nor an output).

- *Locally simulatable MPC is not necessarily topology-hiding:* In locally simulatable MPC, each party is assumed to know their position in the graph (or in other words, the graph class is a singleton). There is no guarantee the parties can correctly run a locally simulatable MPC protocol if they are in an unknown graph setting (and having the parties learn information about the graph to be able to run the protocol would not be topology-hiding).

**From local simulation to execution-obliviousness.** Because the views of two adversarial components  $\mathcal{Z}_1$  and  $\mathcal{Z}_2$  are generated independently, the adversary corrupting the parties cannot tell if  $\mathcal{Z}_1$  and  $\mathcal{Z}_2$  are in fact participating in the same protocol or in different protocols (provided of course they have the same inputs, outputs, and neighborhoods in all these instances).

We are now ready to sketch our topology-hiding broadcast on all graphs, assuming the existence of locally simulatable computation on paths of length  $T = \lambda \cdot n^3$  (which we will instantiate next). Each party  $P_u$  samples  $T$  random permutations on their neighborhood (recall this defines  $2|E| = \sum_{v \in V} \deg_v$  paths, each one visiting each node at least once *w.h.p.*), from which they derive  $2T \cdot \deg_u$  different “path neighborhoods” with the corresponding positions (more precisely,  $2 \deg_u$  of these neighborhoods are as the  $i^{\text{th}}$  node on the path, for each  $i \in [T]$ ). Each party fulfills in parallel their  $2T \cdot \deg_u$  roles in the  $2T \cdot |E|$  parallel executions of locally simulatable OR (the broadcaster always uses the broadcast bit as input and the other parties use 0, in all their roles), one along each path. Their output in each of the protocols is then the broadcast bit.

### 2.1.3 Locally simulatable MPC on a path from OT

By what precedes, topology-hiding broadcast on the class of all graphs can be reduced to a locally simulatable OR on a path. At a high-level, our OR protocol on the path proceeds by recursively emulating a two-party computation (2PC) of an OR. Each party in this top-level 2PC is itself emulated by a 2PC whose two parties are further emulated by a lower-level 2PC, and so on, until we get to 2PCs between two real parties on the path. For a 2PC at any given recursion level, each virtual party is recursively emulated by half of the real parties on the current subset of the path being considered. That is, at the highest level, the first (resp. second) half of the real parties emulate the first (resp. second) virtual party. Then, every  $(1/4)$ -th of the real parties emulate a separate virtual parties at recursion depth 1, and so on, until we reach 2PCs between every pair of neighboring real parties. In a nutshell, local simulatability stems from the fact that each party only sees 2PC messages that come from its direct neighbors. For a more detailed overview of this protocol, we refer the reader to the next subsection.

## 2.2 Technical Overview of the Core Protocol: Locally Simulatable MPC on a Path

We now focus on presenting our main technical contribution of building locally simulatable OR on a path. We first describe our construction (see Figure 3) and then explain the primary ways in which the protocol enables proper topology-hiding emulation and local simulatability.

**Building locally-simulatable OR on a path.** The core step in building our full THB protocol is building a *locally-simulatable* OR protocol on a directed path of length  $\ell = 2^l$ , for some  $l \geq 1$  (each such path will be a random walk, so we can specify its length to be a power of 2). In this setting, each party knows their position on the path (for  $i \in [0, \ell - 1]$ ), and we refer to the party at position  $i$  as  $\tilde{P}_i$ . Given each party  $\tilde{P}_i$ 's input bit  $b_i$ , the protocol outputs  $\bigvee_{i=0}^{\ell-1} b_i$  to every party. When used in the full THB protocol, if  $\tilde{P}_i$  is the broadcaster, then  $b_i = b$ , the broadcast bit; otherwise,  $b_i = 0$ .

In order to compute the OR of their input bits  $b_i$ , the parties emulate recursive (constant-overhead) 2PC computations. At a high-level, the first and second  $\ell/2$  real parties will emulate the first and second virtual parties,  $P_{0,0}$  and  $P_{0,1}$ , respectively, of a 2PC computation. The virtual parties input  $(b_0||b_1||\dots||b_{\ell/2-1})$  and  $(b_{\ell/2}||b_{\ell/2+1}||\dots||b_{\ell-1})$ , respectively, and the 2PC computation outputs to both virtual parties  $\bigvee_{i=0}^{\ell-1} b_i$ . Now, in each round of this 2PC, virtual party  $P_{0,0}$  is emulated recursively via another 2PC between virtual parties  $P_{1,0}$  and  $P_{1,1}$ , which are in turn emulated by, respectively, the first and second  $\ell/4$  real parties recursively (and similarly for virtual party  $P_{0,1}$ ). Virtual party  $P_{0,0}$  is emulated by virtual parties  $P_{1,0}$  and  $P_{1,1}$  as follows:  $P_{1,0}$  and  $P_{1,1}$  combine their inputs  $(b_0||b_1||\dots||b_{\ell/4-1})$  and  $(b_{\ell/4}||b_{\ell/4+1}||\dots||b_{\ell/2-1})$  (via the 2PC) so that  $P_{0,0}$ 's input is emulated by  $x_{0,0} = (b_0||b_1||\dots||b_{\ell/2-1})$ . Similarly,  $P_{1,0}$  and  $P_{1,1}$  each take as input random strings  $\tilde{r}'_{1,0}$  and  $\tilde{r}'_{1,1}$  and combine them (via the 2PC) so that  $P_{0,0}$ 's random tape is emulated by  $\tilde{r}_{0,0} = \tilde{r}'_{1,0} \oplus \tilde{r}'_{1,1}$ . Finally, the 2PC between  $P_{1,0}$  and  $P_{1,1}$  outputs  $P_{0,0}$ 's next message in its 2PC with  $P_{0,1}$  to  $P_{1,1}$ , who then passes it to the first virtual party  $P_{1,2}$  that participates in the 2PC emulating  $P_{0,1}$ . Note that *only* virtual parties  $P_{1,1}$  and  $P_{1,2}$  see and pass to each other the messages for the higher-level 2PC; as we will see later, this is *crucial* to local-simulatability, and topology-hiding in general.

We keep recursively splitting the computation of virtual parties in 2PC's in the recursion, until we reach level  $l - 1$  of the recursion, in which two *real* parties, which are sibling leaves in the recursion tree, compute (many) 2PC's. Again, briefly, the 2PC's that each pair of sibling leaves computes is the emulation of the next message function of the virtual party at the parent in the recursion tree. This virtual party in turn is computing a 2PC with its sibling that emulates the next message function of the virtual party at *their* parent in the recursion tree. We continue up the tree like this, until we reach the original OR between the two largest virtual parties.

For clarity, we depict an example computation with  $\ell = 8$  in Figure 3. We shall first focus on *real* parties  $\tilde{P}_0$  and  $\tilde{P}_1$ . Each party has their respective input bits which we denote as  $b_0$  and  $b_1$ . The parties also sample several random strings for (i) the emulation of virtual party  $P_{0,0}$  and (ii) for the emulation of each 2PC in which virtual party  $P_{1,0}$  participates (one for each of the  $R_{2PC}$  rounds of the root 2PC).

Now, for each round of each 2PC execution that virtual party  $P_{1,0}$  participates in (i.e.,  $R_{2PC}^2$  in total),  $P_{2,0}$  and  $P_{2,1}$  execute their own 2PC to emulate  $P_{1,0}$  in this round. They do so by emulating (via their 2PC execution)  $P_{1,0}$ 's input bits as  $(b_0||b_1)$ ,  $P_{1,0}$ 's input randomness (for emulation of the root 2PC) as  $\tilde{r}'_{1,0} = R_{2,0}[0] \oplus R_{2,1}[0]$ , and  $P_{1,0}$ 's random tape as  $\tilde{r}_{1,0} = \tilde{R}_{2,0}[1] \oplus \tilde{R}_{2,1}[1]$  (where  $\tilde{R}_{2,0}[1], \tilde{R}_{2,1}[1]$  are freshly sampled for each execution in which  $P_{1,0}$  participates).  $P_{2,1}$  then receives  $P_{1,0}$ 's next message as output of this 2PC, and forwards it to  $P_{2,2}$  (who together with  $P_{2,3}$  will emulate  $P_{1,1}$ 's next message). Note that  $P_{2,1}$  will also input to the 2PC  $P_{1,1}$ 's previous messages in its 2PC with  $P_{1,0}$ , which  $P_{2,1}$  receives from  $P_{2,2}$ .

The 2PC's which virtual party  $P_{1,0}$  executes with  $P_{1,1}$  correspond to emulations of the next message function of virtual party  $P_{0,0}$  in the highest level 2PC, which simply computes the OR

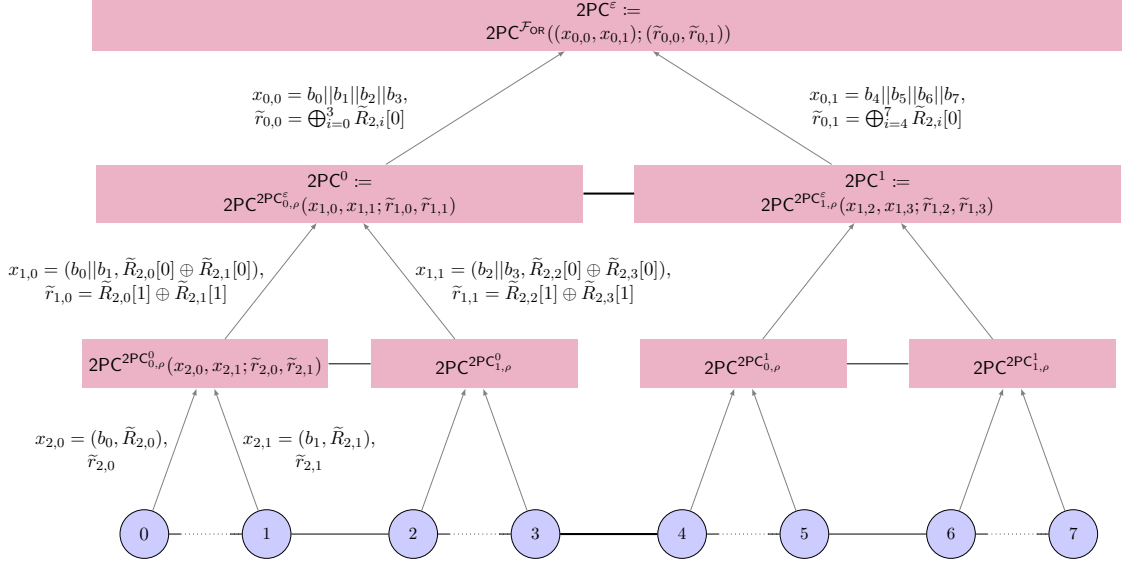


Figure 3: Depiction of the directed path protocol  $\Pi_{\text{dir-path}}$  for a path of length  $\ell = 8$ . Each interior node represents a 2PC which gets its inputs and randomness from its two children. This 2PC computes the next message for virtual party  $P_0$  (resp.  $P_1$ ) in the 2PC at the node's parent by combining the inputs of its two children into the input and randomness of  $P_0$  (resp.  $P_1$ ). This next message is passed from this node to its sibling in the protocol via the two *neighboring real* parties at the rightmost (resp. leftmost) and leftmost (resp. rightmost) leaves of the corresponding subtrees (indicated by horizontal lines of matching thickness).

of  $P_{0,0}$ 's and  $P_{0,1}$ 's input bits.  $P_{1,0}$  and  $P_{1,1}$  emulate (via these 2PC executions)  $P_{0,0}$ 's input bits as  $(b_0||b_1||b_2||b_3)$ , and  $P_{0,0}$ 's random tape as  $\tilde{r}_{0,0} = \tilde{r}'_{1,0} \oplus \tilde{r}'_{1,1}$ . Again, recall that, recursively,  $P_{1,0}$  (resp.  $P_{1,1}$ ) was emulated by  $P_{2,0}$  and  $P_{2,1}$  (resp.  $P_{2,2}$  and  $P_{2,3}$ ) so that its input bits were  $(b_0||b_1)$  (resp.  $(b_2||b_3)$ ) and  $\tilde{r}'_{1,0} = \tilde{R}_{2,0}[0] \oplus \tilde{R}_{2,1}[0]$  (resp.  $\tilde{r}'_{1,1} = \tilde{R}_{2,2}[0] \oplus \tilde{R}_{2,3}[0]$ ). So, when  $P_{1,1}$  computes its output, it will be the next message of  $P_{0,0}$  in its 2PC computation with  $P_{1,0}$  of the OR functionality, with input  $x_{0,0} = b_0||b_1||b_2||b_3$  and random tape  $\tilde{r}_{0,0} = \bigoplus_{i=0}^3 \tilde{R}_{2,i}[0]$ . This output will then be recursively passed down (via another 2PC) to  $P_{2,3}$ , who will then pass it to virtual party  $P_{0,1}$  via real party  $P_{2,4}$ .  $P_{0,1}$ 's messages in the 2PC with  $P_{0,0}$  will be similarly recursively emulated so that when  $P_{0,0}$  and  $P_{0,1}$  finally compute their outputs in the highest-level OR 2PC execution, they will be recursively passed down to each  $P_{1,i}$ , and then again to each  $P_{2,i}$  so that finally, all parties  $\tilde{P}_i$  receive  $\bigvee_{i=0}^7 b_i$ .

Finally, note that the recursion depth is just  $l = \log_2(\ell)$ . Moreover, when the 2PC is implemented with a constant round 2PC with constant computational overhead, we can see that the round complexity grows multiplicatively in the recursion depth, i.e.  $O(1)^l = \text{poly}(\ell)$ , and moreover the total computational complexity (and hence communication complexity) is just  $O(\dots O(O(1) + \text{poly}(\lambda)) + \text{poly}(\lambda) \dots) + \text{poly}(\lambda) = \text{poly}(\ell, \lambda)$ .

**Enablers for Proper Topology-Hiding Emulation and Local Simulatability.** There are a few main ways in which this protocol enables proper topology-hiding emulation and local simulatability.

bility. First, 2PC messages at any depth of the recursion are only output and passed between real parties that are neighbors on the path. This is important since if this were not true, and (random-looking, and thus unique w.h.p.) messages were passed between real parties several edges away from each other, then as noted previously, these parties would know that they participate in the same execution, and thus local simulation would not be possible. This is the reason why our path protocol uses recursive 2PC’s, as opposed to, e.g., 3PC’s, as doing so would require real parties to pass messages to other real parties that are not their neighbors, thus revealing information about the topology (recall that we work in the  $KT_0$  model, so parties should not know if they have a neighbor in common).

Second, virtual parties’ random tapes are collectively emulated by each real party of which they consist. So, even if the party at the “edge” of a virtual party that sees the 2PC messages sent by the virtual party they are helping to emulate is corrupted, if at least one of the other real parties in the virtual party is uncorrupted, then this 2PC message reveals nothing about the uncorrupted parties’ inputs. This is because the uncorrupted parties mix in their own fresh randomness to compute the random tape of the virtual party so that the 2PC messages are generated with randomness that looks fresh and independent to the adversary. So, by the security of the 2PC, these messages reveal nothing about the virtual party’s input (and thus nothing about the uncorrupted real parties’ inputs).

Finally, since we compute an OR amongst all parties, we can simulate virtual parties’ views with only partial information. Simulation using generic 2PC seems challenging at a first glance, since in the 2PC in which a corrupted real party is helping to emulate a virtual party, it may receive 2PC messages from the other virtual party in the higher-level 2PC. This happens even if some of the other real parties of which the emulated virtual party consists are not corrupted. We are thus faced with using generic 2PC simulators only with partial information on the input (and output) of the corresponding virtual party. However, since we compute the OR functionality, and based on the output OR’d bit  $b$  and the fact that every real party mixes in their own independent randomness for the emulation of virtual parties, our local simulators can actually *fill in the gaps* of the uncorrupted parties. That is, if  $b = 0$ , then our simulator can simply fill in the uncorrupted parties’ inputs as 0 and sample fresh randomness for them, which will be a perfect simulation. Even if  $b = 1$ , because of the 2PC security of computing ORs, our simulator can simply simulate as if *all* of the uncorrupted parties’ inputs were 1. Although this will not be true for the THB protocol itself, it can be true for computing recursive ORs, and thus we leverage this along with 2PC security for our proof.

**Generalizing to “Efficiently Invertible from Local Information” functionalities.** We just noted that the fact that our path protocol computes an OR is crucial to local simulatability. The important part, however, was that from a subset of parties’ input bits and the output bit, one can efficiently compute all other parties’ inputs (0’s if the output is 0; 1’s if the output is 1). We can further generalize this strategy to all functionalities  $\mathcal{F}$  such that given a subset of parties’ inputs and outputs, there exists an “inverse” algorithm that computes possible inputs of the other parties that are consistent with the original parties’ outputs. We call such functionalities *efficiently invertible from local information*. Other examples of such functionalities include private set intersection, private set union, and more. However, we do note that there are some efficiently computable functionalities that nonetheless are not efficiently invertible from local information; for example, leakage resilient one-way functions. Unfortunately, we cannot extend the strategy to such

functionalities.

Now recall that we use secure OR to eventually build our THB protocol, which in turn can be generically composed with any secure MPC protocol to get full-fledged THC (see Section 6). However, we note that if the eventual THC computes a functionality that is efficiently invertible from local information, our path protocol can just directly (and thus more efficiently) be used to compute the THC, without going through the THB + MPC composition.

### 2.3 A Note on the Differences with [MOR15]

There are three main differences between our core recursive protocol and the recursive protocol of [MOR15]. First, our recursion is much more efficient—with every additional depth of recursion, we double the number of parties, while [MOR15] only adds two more parties (on paths). In more detail, the base protocol of [MOR15] starts off with each node  $v$  computing several MPCs with its local neighborhood,  $\mathcal{N}_G[v] = \{v\} \cup \{u : u \text{ has an edge with } v\}$ . This protocol is only secure if every node  $v$ 's neighborhood has at least one uncorrupted party. Then, to increase security, they repeatedly construct new protocols based on the previous constructed protocol, where each node  $v$ 's role in the previous protocol is instead recursively computed using MPCs amongst its neighborhood; each message that is sent to  $v$  from some neighbor  $u$  (resp. from  $v$  to  $u$ ) in the previous protocol is output to  $v$  (resp.  $u$ ) via an MPC computed by  $u$ 's (resp.  $v$ 's) neighborhood, encrypted under a key that is secret shared amongst  $v$ 's (resp.  $u$ 's) neighborhood. So, the  $(k+1)$ -st protocol recursively computes a given node  $v$ 's role in the base protocol using  $v$ 's  $k$ -neighborhood  $\mathcal{N}_G^k[v]$ , where  $\mathcal{N}_G^k[v] = \bigcup_{u \in \mathcal{N}_G^{k-1}[v]} \mathcal{N}_G[u]$ . Security is achieved if every node  $v$ 's  $(k+1)$ -neighborhood has at least one uncorrupted party. This inefficient style of recursion is indeed why [MOR15] can only perform THC on graphs of constant degree and logarithmic diameter, even using constant round MPC with constant computational overhead: Each additional level of recursion increases the overhead of the protocol by a factor of  $\deg(v)$ . However, in our path protocol, each new level of recursion introduces a new virtual 2PC, which doubles the number of parties involved in the protocol. So, with logarithmically many recursion levels, we can cover the whole path, while retaining efficiency roughly proportional to  $c^{\log(\ell)}$ , where  $\ell$  is the length of the path and  $c$  is some constant representing the overhead of the 2PC protocol.

The second difference comes from the fact that our novel use of more efficient recursion introduces the simulation challenge discussed earlier: We must use generic 2PC simulators with only partial knowledge of inputs (and outputs). This comes from the fact that 2PC messages amongst virtual parties are passed between real parties in the protocol. However, such message passing does not occur in the protocol of [MOR15] and indeed its simulation is as follows: If a node  $v$ 's neighborhood is not fully corrupted, then simulate  $v$ 's view based on its output messages being encryptions of 0 (this is valid, since the adversary will not know the corresponding secret key). Otherwise, simulate  $v$ 's view using the simulator from some lower-level protocol: Let  $k$  be the largest integer such that  $\mathcal{N}_G^k[v]$  is corrupted. Now note that the highest-level protocol just has this  $k$ -neighborhood of  $v$  compute  $v$ 's role in some lower-level protocol and then gives  $v$  the corresponding output in the last round. Thus, the higher-level protocol's simulator can recover the corresponding entire input and output of  $v$  in that lower-level protocol, and therefore simulate  $v$ 's view properly based on the lower-level simulator. Therefore, the simulation of [MOR15] does not need to handle the same technical challenges as ours.

The final difference is that the protocol of [MOR15] only works in the  $\text{KT}_1$  model, since MPCs between neighborhoods are computed, so messages are passed between parties two edges away from



each other. However, our protocol works in the  $\text{KT}_0$  model, since messages are only passed directly between neighbors.

### 3 Preliminaries

**Notations.** For  $m < n \in \mathbb{N}$  let  $[n] = \{1, \dots, n\}$  and  $[m, n] = \{m, m + 1, \dots, n\}$ . In our protocols we sometimes denote by  $B$  an upper bound on the number of participating parties. The security parameter is denoted by  $\lambda$ . We will use 0-indexing for many of our definitions and protocols. We also make use of dictionaries in our protocols. For a dictionary  $D$ ,  $D[:x]$  results in a new dictionary  $D'$  consisting of elements 0 through  $x$  of  $D$ ; i.e., for  $i \in [0, x]$ ,  $D'[i] = D[i]$ , but for  $i > x$ ,  $D'[i] = \perp$ . Finally, we let  $\|_{j=i}^n x_j = x_i \| x_{i+1} \| \dots \| x_n$

**Graph notations and properties.** A graph  $G = (V, E)$  is a set  $V$  of vertices and a set  $E$  of edges, each of which is an unordered pair  $\{v, w\}$  of distinct vertices. A graph is *directed* if its edges are instead ordered pairs  $(v, w)$  of distinct vertices. The (*open*) *neighbourhood* of a vertex  $v$  in an undirected graph  $G$ , denoted  $\mathcal{N}_G(v)$ , is the set of vertices sharing an edge with  $v$  in  $G$ . The *closed neighbourhood* of  $v$  in  $G$  is in turn defined by  $\mathcal{N}_G[v] := \mathcal{N}_G(v) \cup \{v\}$ .

#### 3.1 Topology-Hiding Computation (THC)

There are two notions of topology-hiding computation in the literature: game-based and simulation-based [MOR15]. Since we introduce a feasibility result, we use a stronger simulation-based definition.

**UC framework.** The simulation-based definition is defined in the UC framework of [Can00]. We will consider computationally bounded, static, and semi-honest adversaries and environments.

**Neighbourhood Models.** In this work, we unify the neighbourhood models of past THC definitions in the literature (for an illustration we refer to Fig. 4). To simplify the notation, we will consider that  $P_v$  in some protocol is associated with node  $v$  in the underlying graph. Typically, THC functionalities are realized in the  $\mathcal{F}_{\text{graph}}^{\mathcal{G}}$ -hybrid model, where  $\mathcal{F}_{\text{graph}}^{\mathcal{G}}$  is some functionality that allows parties to communicate with their neighbors in the graph. Many works have used the model of [MOR15], wherein  $\mathcal{F}_{\text{graph}}^{\mathcal{G}}$  informs every party  $P_v$  of their local neighbourhood by indeed sending  $\mathcal{N}_G(v)$  directly to them, and  $\mathcal{F}_{\text{graph}}^{\mathcal{G}}$  thereafter facilitates communication from  $P_v$  to some other node  $u$ , only if  $u$  is indeed a neighbor of  $v$ . However, [ALM17] instead has  $\mathcal{F}_{\text{graph}}^{\mathcal{G}}$  first sample a random injective function  $f: E \rightarrow [n^2]$ , labeling each edge with a random (unique) element from  $[n^2]$ . Next,  $\mathcal{F}_{\text{graph}}^{\mathcal{G}}$  informs every party  $P_v$  of their local neighbourhood by instead sending them the set of edge labels  $L_v := \{f((u, v)): (u, v) \in E\}$ .  $\mathcal{F}_{\text{graph}}^{\mathcal{G}}$  thereafter facilitates communication from  $P_v$  along some edge with label  $l$ , only if  $l$  corresponds to some edge  $(v, u) \in E$  according to  $f$ .

We refer to these two notions according to the terminology of [AGPV88], who define the **Knowledge Till Radius  $\sigma$  Model** ( $\text{KT}_\sigma$ ). These two worlds are illustrated in Figure 4.  $\text{KT}_1$  is called the ‘Common Neighbours’ model, and refers to the [MOR15] world. Indeed, in this world, parties are given the identities of their neighbours, so that two colluding parties that each have an edge to a common party know that this is in fact the case.  $\text{KT}_0$  is called the ‘Pseudonymous

Neighbours’ model, and refers to the [ALM17] world. In this world, parties are only given the random (unique) identities of the *edges* corresponding to their neighbourhood, as described above, but not the actual identities of the parties with which they share these edges. So, if two colluding parties each have an edge to a common party, their respective edges will have different labelings and thus will not tell them if they indeed share this common neighbour.

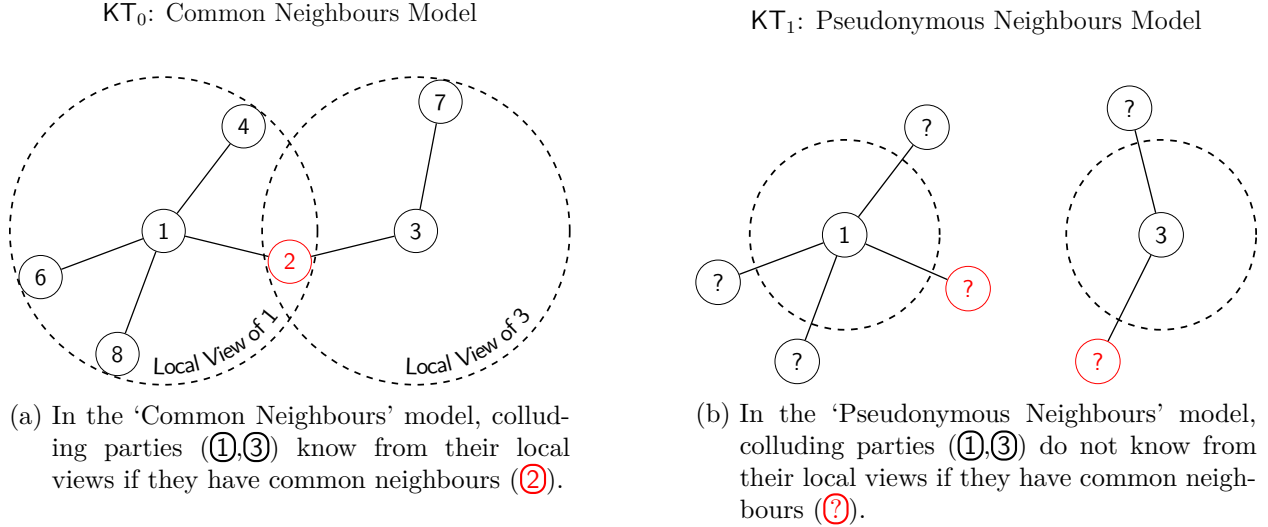


Figure 4: Differing views of parties in KT<sub>0</sub> and KT<sub>1</sub>.

### 3.1.1 Simulation-Based THC.

Now we are ready to introduce our simulation-based topology-hiding computation definition. The real-world protocol is defined in a model where all communication is transmitted via the functionality  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}_\sigma}$  (described in Figure 5). The functionality is parameterised by a family of graphs  $\mathcal{G}$ , representing all possible network topologies (aka communication graphs) that the protocol supports. It is also parameterised by the neighbourhood model  $\text{KT}_\sigma$ , for  $\sigma \in \{0, 1\}$ . We implicitly assume that every node in a graph is associated with a specific *party identifier*, *pid*.

Initially, before the protocol begins,  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}_\sigma}$  receives the network communication graph  $G$  from a special graph party  $P_{\text{graph}}$  and makes sure that  $G \in \mathcal{G}$ . Then, if  $\sigma = 0$ , it samples a random injective function  $f: E \rightarrow [n^2]$ , labeling each edge with an element from  $[n^2]$ , and gives each party  $P_v$  with  $v \in V$  the edge labels according to its local neighbor-set. Next, during the protocol’s execution, whenever party  $P_v$  wishes to send a message  $m$  along edge  $l$ , it sends  $(l, m)$  to the functionality; the functionality first checks if there is  $(v, w) \in E$  such that  $f(v, w) = l$ , and if so delivers  $(l, m)$  to  $P_w$ . Otherwise, if  $\sigma = 1$ , it simply provides to each party  $P_v$  with  $v \in V$  its local neighbor-set. Next, during the protocol’s execution, whenever party  $P_v$  wishes to send a message  $m$  to party  $P_w$ , it sends  $(v, w, m)$  to the functionality; the functionality verifies that the edge  $(v, w)$  is indeed in the graph, and if so delivers  $(v, w, m)$  to  $P_w$ .

Note that if all the graphs in  $\mathcal{G}$  have exactly  $n$  nodes, then the exact number of participants is known to all and need not be kept hidden. In this case, defining the ideal functionality and constructing protocols becomes a simpler task. However, if there exist graphs in  $\mathcal{G}$  that contain

a *different* number of nodes, then the model must support functionalities and protocols that only know an *upper bound* on the number of participants. In the latter case, the actual number of participating parties must be kept hidden.

Given a class of graphs  $\mathcal{G}$  with an upper bound  $n$  on the number of parties, we define a protocol  $\pi$  with respect to  $\mathcal{G}$  as a set of  $n$  PPT interactive Turing machines (ITMs)  $(P_1, \dots, P_n)$  (the parties), where any subset of them may be activated with (potentially empty) inputs. Only the parties that have been activated participate in the protocol, send messages to one another (via  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}\sigma}$ ), and produce output.

### Functionality $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}\sigma}$

The functionality  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}\sigma}$  is parametrized by a graph class  $\mathcal{G}$  and neighbourhood model  $\text{KT}\sigma$ ; let  $n$  be the maximum number of nodes in any graph in  $\mathcal{G}$ .  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}\sigma}$  interacts with a special graph party  $P_{\text{graph}}$  and (a subset of the) parties  $P_1, \dots, P_n$  (to be defined by the graph received from  $P_{\text{graph}}$ ) as follows.

#### Initialization Phase:

**Input:**  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}\sigma}$  waits to receive the graph  $G = (V, E)$  from  $P_{\text{graph}}$ . If  $G \notin \mathcal{G}$ , abort. If  $\sigma = 0$ ,  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}\sigma}$  samples a random injective function  $f: E \rightarrow [n^2]$ , labeling each edge with an element from  $[n^2]$ .

**Output:** Upon receiving an initialization message from  $P_v$ ,  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}\sigma}$  verifies that  $v \in V$ , and if so sends the following to  $P_v$ :

$$\begin{cases} \mathcal{N}_G(v) & \text{if } \sigma = 1 \text{ ("in } \text{KT}_1 \text{")} \\ L_v := \{f((u, v)) : (u, v) \in E\} & \text{if } \sigma = 0 \text{ ("in } \text{KT}_0 \text{")} \end{cases}$$

#### Communication Phase:

##### Input:

- If  $\sigma = 1$ :  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}_1}$  receives from a party  $P_v$  a destination/data pair  $(w, m)$  where  $w \in \mathcal{N}_G(v)$  and  $m$  is the message  $P_v$  wants to send to  $P_w$ . (If  $v, w \notin V$ , or if  $w$  is not a neighboring vertex of  $v$ ,  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}_0}$  ignores this input.)
- If  $\sigma = 0$ :  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}_0}$  receives from a party  $P_v$  a destination/data pair  $(l, m)$  where  $f(v, w) = l \in L_v$  indicates to  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}_0}$  the neighbour  $w$ , and  $m$  is the message  $P_v$  wants to send to  $P_w$ . (If  $v \notin V$  or  $\nexists (v, w) \in E : f(v, w) = l$ ,  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}_1}$  ignores this input.)

##### Output:

- If  $\sigma = 1$ :  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}_1}$  gives output  $(v, m)$  to  $P_w$  indicating that  $P_v$  sent the message  $m$  to  $P_w$ .
- If  $\sigma = 0$ :  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}_0}$  gives output  $(l, m)$  to  $P_w$ , where  $f(v, w) = l$ , indicating that the neighbor on edge  $l$  sent the message  $m$  to  $P_w$ .

Figure 5: The communication graph functionality (unified definition for  $\text{KT}_0$  and  $\text{KT}_1$ ).

An ideal-model computation of a functionality  $\mathcal{F}$  is augmented to provide the corrupted parties with the information that is leaked about the graph; namely, every corrupted (dummy) party should learn its local neighbourhood information (in  $\text{KT}_0$  or  $\text{KT}_1$ , respectively). Note that the functionality  $\mathcal{F}$  can be completely agnostic about the actual graph that is used, and even about the family  $\mathcal{G}$ . To augment  $\mathcal{F}$  in a generic way, we define the wrapper-functionality  $\mathcal{W}_{\text{graph-info}}^{\mathcal{G}, \text{KT}_\sigma}(\mathcal{F})$ , that runs internally a copy of the functionality  $\mathcal{F}$ . The wrapper  $\mathcal{W}_{\text{graph-info}}^{\mathcal{G}, \text{KT}_\sigma}(\cdot)$  acts as a shell that is responsible to provide the relevant leakage to the corrupted parties; the original functionality  $\mathcal{F}$  is the core that is responsible for the actual ideal computation.

More specifically, the wrapper receives the graph  $G = (V, E)$  from the graph party  $\text{P}_{\text{graph}}$ , makes sure that  $G \in \mathcal{G}$ , and sends a special initialization message containing  $G$  to  $\mathcal{F}$ . (If the functionality  $\mathcal{F}$  does not depend on the communication graph, it can ignore this message.) The wrapper then proceeds to process messages as follows: Upon receiving an initialization message from a party  $\text{P}_v$ , responds with its local neighbourhood information (just like  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}_\sigma}$ ). All other input messages from a party  $\text{P}_v$  are forwarded to  $\mathcal{F}$  and every message from  $\mathcal{F}$  to a party  $\text{P}_v$  is delivered to its recipient.

### Wrapper Functionality $\mathcal{W}_{\text{graph-info}}^{\mathcal{G}, \text{KT}_\sigma}(\mathcal{F})$

The wrapper functionality  $\mathcal{W}_{\text{graph-info}}^{\mathcal{G}, \text{KT}_\sigma}(\mathcal{F})$  is parametrized by a graph class  $\mathcal{G}$  and neighbourhood model  $\text{KT}_\sigma$ ; let  $n$  be the maximum number of nodes in any graph in  $\mathcal{G}$ .  $\mathcal{W}_{\text{graph-info}}^{\mathcal{G}, \text{KT}_\sigma}(\mathcal{F})$  internally runs a copy of  $\mathcal{F}$  and interacts with a special graph party  $\text{P}_{\text{graph}}$  and (a subset of the) parties  $\text{P}_1, \dots, \text{P}_n$  (to be defined by the graph received from  $\text{P}_{\text{graph}}$ ) as follows.

#### Initialization Phase:

**Input:**  $\mathcal{W}_{\text{graph-info}}^{\mathcal{G}, \text{KT}_\sigma}(\mathcal{F})$  waits to receive the graph  $G = (V, E)$  from  $\text{P}_{\text{graph}}$ . If  $G \notin \mathcal{G}$ , abort.  
If  $\sigma = 0$ ,  $\mathcal{W}_{\text{graph-info}}^{\mathcal{G}, \text{KT}_\sigma}(\mathcal{F})$  samples a random injective function  $f: E \rightarrow [n^2]$ , labeling each edge with an element from  $[n^2]$ .

**Outputs:** Upon receiving an initialization message from  $\text{P}_v$ ,  $\mathcal{W}_{\text{graph-info}}^{\mathcal{G}, \text{KT}_\sigma}(\mathcal{F})$  verifies that  $v \in V$ , and if so sends the following to  $\text{P}_v$ :

$$\begin{cases} \mathcal{N}_G(v) & \text{if } \sigma = 1 \text{ ("in } \text{KT}_1 \text{")} \\ L_v := \{f((u, v)): (u, v) \in E\} & \text{if } \sigma = 0 \text{ ("in } \text{KT}_0 \text{")} \end{cases}$$

#### Communication Phase:

**Input:**  $\mathcal{W}_{\text{graph-info}}^{\mathcal{G}, \text{KT}_\sigma}(\mathcal{F})$  forwards every message it receives to  $\mathcal{F}$ .

**Output:** Whenever  $\mathcal{F}$  sends a message,  $\mathcal{W}_{\text{graph-info}}^{\mathcal{G}, \text{KT}_\sigma}(\mathcal{F})$  forwards the message to its intended recipient.

Figure 6: The graph-information wrapper functionality (unified definition for  $\text{KT}_0$  and  $\text{KT}_1$ ).

Note that formally, the set of all possible parties  $V^*$  is fixed in advance. To represent a graph  $G' = (V', E')$  where  $V' \subseteq V^*$  is a subset of the parties, we use the graph  $G = (V^*, E')$ , where all vertices  $v \in V^* \setminus V'$  have degree 0.

**Definition 3.1** (Topology-hiding computation). *We say that a protocol  $\pi$  securely realizes a functionality  $\mathcal{F}$  in a topology-hiding manner with respect to  $\mathcal{G}$  tolerating a semi-honest adversary corrupting  $t$  parties if  $\pi$  securely realizes  $\mathcal{W}_{\text{graph-info}}^{\mathcal{G}, \text{KT}_0}(\mathcal{F})$  in the  $\mathcal{F}_{\text{graph}}^{\mathcal{G}, \text{KT}_0}$ -hybrid model tolerating a semi-honest adversary corrupting  $t$  parties.*

**Broadcast.** In this work we will focus on topology-hiding computation of the *broadcast* functionality (see Figure 7), where a designated and publicly known party, named *the broadcaster*, starts with an input value  $m$ . Our broadcast functionality guarantees that every party that is connected to the broadcaster in the communication graph receives the message  $m$  as output. In this paper, we assume the communication graphs are always connected. However, the broadcaster may not be participating, in which case it is represented as a degree-0 node in the communication graph (and all the participating nodes are in a separate connected component.)

Parties that are not connected to the broadcaster receive a message that is supplied by the adversary (we can consider stronger versions of broadcast, but this simplifies the proofs).

We denote the broadcast functionality where the broadcaster is  $P_i$  by  $\mathcal{F}_{\text{bc}}(P_i)$ .

**Functionality  $\mathcal{F}_{\text{bc}}(P_i)$**

The broadcast functionality  $\mathcal{F}_{\text{bc}}(P_i)$  is parametrized by the broadcaster  $P_i$  and proceeds as follows.

**Initialization:** The functionality receives the communication graph  $G$  from the wrapper  $\mathcal{W}_{\text{graph-info}}$ .

**Input:** Record the input message  $m \in \{0, 1\}$  sent by the broadcaster  $P_i$ .

**Output:** Send the output  $m$  to every party that is in the same connected component as  $P_i$  in  $G$ . For every other party in  $G$ , the output delivered to that party is supplied by the adversary.

Figure 7: The broadcast functionality

**Definition 3.2** ( $t$ -THB). *Let  $\mathcal{G}$  be a family of graphs and let  $t$  be an integer. A protocol  $\pi$  is a  $t$ -THB protocol with respect to  $\mathcal{G}$  if  $\pi(P_v)$  securely realizes  $\mathcal{F}_{\text{bc}}(P_v)$  in a topology-hiding manner with respect to  $\mathcal{G}$ , for every  $P_v$ , tolerating a semi-honest adversary corrupting  $t$  parties.*

## 3.2 Constant-Overhead Two-Party Computation for Semi-Honest Adversaries

**Definition 3.3** (Stateless Two-Party Computation Syntax). A  $R_{2PC}$ -round Stateless Two-Party Computation (2PC) protocol  $2PC^{\mathcal{F}}(x_0, x_1; r_0, r_1) := (2PC_{0,i}^{\mathcal{F}}, 2PC_{1,i}^{\mathcal{F}})_{i \in [0, R_{2PC}-1]}$  for given functionality  $\mathcal{F}$  is described by two parties,  $P_0$  and  $P_1$ , with respective inputs  $x_0, x_1$  and respective randomness  $r_0, r_1$  that use PPT algorithms  $2PC_{0,i}^{\mathcal{F}}(x_0, \{m_{1,j}\}_{j < i}; r_0)$  (resp.  $2PC_{1,i}^{\mathcal{F}}(x_1, \{m_{0,j}\}_{j \leq i}; r_1)$ ) to compute  $P_0$ 's (resp.  $P_1$ 's)  $i$ -th round message of the protocol,  $m_{0,i}$  (resp.  $m_{1,i}$ ), taking as input  $P_0$ 's 2PC input  $x_0$  and the  $j$ -th round messages of  $P_1$  for  $j < i$ , and using  $P_0$ 's 2PC randomness  $r_0$  (resp.  $P_1$ 's 2PC input  $x_1$  and the  $j$ -th round messages of  $P_0$  for  $j \leq i$ , and using  $P_1$ 's 2PC randomness  $r_1$ ). Algorithm  $2PC_{0, R_{2PC}-1}^{\mathcal{F}}(x_0, \{m_{1,j}\}_{j < R_{2PC}-1}; r_0)$  (resp.  $2PC_{1, R_{2PC}-1}^{\mathcal{F}}(x_1, \{m_{0,j}\}_{j \leq R_{2PC}-1}; r_1)$ ) additionally gives output  $y_0$  (resp. only gives output  $y_1$ ).

We will additionally use the notation  $2PC_{i, < \rho}^{\mathcal{F}}(x_0, x_1; r_0, r_1)$  to represent the first  $\rho$  messages that party  $i$  receives from party  $1 - i$  on inputs  $x_0, x_1$  and randomness  $r_0, r_1$ , respectively.

**Constant-overhead constant-round 2PC.** For this work, we need to use a 2PC with constant overhead and constant round complexity. More precisely, we require that 2PC satisfies the following properties: First, for any given functionality  $\mathcal{F}$  and the corresponding circuit  $\mathcal{C}_{\mathcal{F}}$  that computes it, 2PC has computational (and thus also communication) overhead  $O(|\mathcal{C}_{\mathcal{F}}|) + \text{poly}(\lambda)$ , where  $|\mathcal{C}_{\mathcal{F}}|$  is the size of the circuit, i.e., the number of gates it has. Second, we require the number of rounds  $R_{2PC}$  to be constant.

### 3.2.1 Semi-Honest Security

We follow the standard real/ideal world paradigm [Gol04]. Consider two parties  $P_0, P_1$ , with inputs  $x_0, x_1$  and randomness  $r_0, r_1$ , respectively, who execute protocol  $2PC^{\mathcal{F}}$  to evaluate given functionality  $\mathcal{F}$  on their joint inputs, using their joint randomness. The security of protocol  $2PC^{\mathcal{F}}$  is defined by comparing the real-world execution of the protocol with an ideal-world evaluation of  $\mathcal{F}$  by a trusted party (ideal functionality). Informally, it is required that for every adversary  $\mathcal{A}$  that chooses to *statically* corrupt either  $P_0$  or  $P_1$  at the beginning of the real execution of the protocol, there exists an adversary  $\text{Sim}$ , also referred to as the simulator, which can achieve the same effect in the ideal-world. For simplicity, we will refer to an adversary that chooses to corrupt  $P_i$  as  $\mathcal{A}_i$ , and similarly  $\text{Sim}_i$  for the corresponding simulator, but the definition holds for all adversaries  $\mathcal{A}_0$  and  $\mathcal{A}_1$ . We now formally describe the security definition.

**Real Execution.** In the real execution,  $2PC^{\mathcal{F}}$  is executed in the presence of an adversary  $\mathcal{A}_i$ . The *view* of the first (resp. second) party during an execution of  $2PC^{\mathcal{F}}(x_0, x_1; r_0, r_1)$  denoted  $\text{VIEW}_0^{2PC^{\mathcal{F}}}(x_0, x_1)$  (resp.  $\text{VIEW}_1^{2PC^{\mathcal{F}}}(x_0, x_1)$ ) is  $(x_0, r_0, \{m_{1,i}\}_{i \in [0, R_{2PC}-1]})$  (resp.  $(x_1, r_1, \{m_{0,i}\}_{i \in [0, R_{2PC}-1]})$ ). The *output* of the first (resp. second) party after an execution of  $2PC^{\mathcal{F}}(x_0, x_1; r_0, r_1)$  denoted  $\text{OUTPUT}_0^{2PC^{\mathcal{F}}}(x_0, x_1)$  (resp.  $\text{OUTPUT}_1^{2PC^{\mathcal{F}}}(x_0, x_1)$ ) is implicit in the party's own view of the execution and  $\text{OUTPUT}^{2PC^{\mathcal{F}}}(x_0, x_1) = (\text{OUTPUT}_0^{2PC^{\mathcal{F}}}(x_0, x_1), \text{OUTPUT}_1^{2PC^{\mathcal{F}}}(x_0, x_1))$ . The execution of  $2PC^{\mathcal{F}}$  in the presence of  $\mathcal{A}_i$  on input  $(x_0, x_1)$  and auxiliary input  $z$ , denoted  $\text{REAL}_{2PC^{\mathcal{F}}, \mathcal{A}_i(z)}(x_0, x_1)$ , is defined as  $(\text{OUTPUT}^{2PC^{\mathcal{F}}}(x_0, x_1), \mathcal{A}_i(\text{VIEW}_i^{2PC^{\mathcal{F}}}(x_0, x_1), z))$ .



**Ideal Execution.** In the ideal execution, the two parties and an ideal world adversary  $\text{Sim}_i$  interact with a trusted party (ideal functionality). The ideal execution proceeds as follows: Both parties send their respective inputs  $x_0, x_1$  to the trusted party, who computes  $\mathcal{F}(x_0, x_1)$  and sends the respective outputs  $\mathcal{F}_0(x_0, x_1), \mathcal{F}_1(x_0, x_1)$  to the parties. The execution of  $\mathcal{F}$  in the presence of  $\text{Sim}_i$  in the ideal model on input  $(x_0, x_1)$  and auxiliary input  $z$ , denoted  $\text{IDEAL}_{\mathcal{F}, \text{Sim}_i(z)}(x_0, x_1)$ , is defined as  $(\mathcal{F}(x_0, x_1), \text{Sim}_i(x_i, \mathcal{F}_i(x_0, x_1), z))$ .

**Definition 3.4** (Static, Semi-Honest Two-Party Computation Security). *Protocol  $2\text{PC}^{\mathcal{F}}$  securely computes  $\mathcal{F}$  in the static, semi-honest model if for every PPT adversary  $\mathcal{A}_i$ , there exists a PPT simulator  $\text{Sim}_i$  such that:*

$$\{\text{REAL}_{2\text{PC}^{\mathcal{F}}, \mathcal{A}_i(z)}(x_0, x_1)\}_{x_0, x_1, z \in \{0,1\}^*} \approx_c \{\text{IDEAL}_{\mathcal{F}, \text{Sim}_i(z)}(x_0, x_1)\}_{x_0, x_1, z \in \{0,1\}^*}$$

### 3.3 Efficiently Invertible from Local Information Functionalities

We now define a special property for functionalities needed to prove local simulatability of our path protocol. We first define *input consistency*:

**Definition 3.5** (Input Consistency). *Given some subset  $J \subseteq [0, n-1]$  and corresponding inputs and outputs  $\{(x_j, y_j)\}_{j \in J} \subseteq \{(x_0, y_0), \dots, (x_{n-1}, y_{n-1})\}$  of a functionality  $\mathcal{F}$ , a set of inputs  $\{x'_j\}_{j \in [0, n-1] \setminus J}$  is input-consistent with  $\{(x_j, y_j)\}_{j \in J}$  if  $\mathcal{F}$  on input  $\{x_j\}_{j \in J}$  and  $\{x'_j\}_{j \in [0, n-1] \setminus J}$  outputs  $y'_0, \dots, y'_{n-1}$  such that  $\{y_j\}_{j \in J} = \{y'_j\}_{j \in J}$ .*

**Definition 3.6** (Efficiently Invertible from Local Information Functionalities). *A functionality  $(y_0, \dots, y_{n-1}) = \mathcal{F}(x_0, \dots, x_{n-1})$  is efficiently invertible from local information if given some subset  $J \subseteq [0, n-1]$  and corresponding inputs and outputs of  $\mathcal{F}$ ,  $\{(x_j, y_j)\}_{j \in J} \subseteq \{(x_0, y_0), \dots, (x_{n-1}, y_{n-1})\}$ , there exists a PPT algorithm  $\{(x'_j)\}_{[0, n-1] \setminus J} = \mathcal{F}^{-1}(\{(x_j, y_j)\}_{j \in J})$  such that  $\{(x'_j)\}_{[0, n-1] \setminus J}$  are input-consistent with  $\{(x_j, y_j)\}_{j \in J}$  for  $\mathcal{F}$ .*

It can easily be seen that the OR functionality  $\mathcal{F}_{\text{OR}}(b_0, \dots, b_{n-1}) = \bigvee_{j=0}^{n-1} b_j$  (where all parties receive this same output) is efficiently invertible from local information. If the output is 0, one immediately knows all unknown inputs must be 0; if the output is 1, one can imagine that all unknown inputs are 1, as this still results in the same output, 1.

Now, we prove a simple lemma showing that input consistency is symmetric:

**Lemma 3.7.** *Given some subset  $J \subseteq [0, n-1]$  and corresponding inputs and outputs of some functionality  $\mathcal{F}$ ,  $\{(x_j, y_j)\}_{j \in J} \subseteq \{(x_0, y_0), \dots, (x_{n-1}, y_{n-1})\}$ , and input-consistent  $\{(x'_j)\}_{[0, n-1] \setminus J}$ , if  $\mathcal{F}$  on input  $\{x_j\}_{j \in J}$  and  $\{x'_j\}_{j \in [0, n-1] \setminus J}$  outputs  $y'_0, \dots, y'_{n-1}$ , then  $\{x_j\}_{j \in J}$  is input-consistent with  $\{(x'_j, y'_j)\}_{j \in [0, n-1] \setminus J}$ .*

*Proof.* This is immediate based on the fact that  $\mathcal{F}$  on input  $\{x_j\}_{j \in J}$  and  $\{x'_j\}_{j \in [0, n-1] \setminus J}$  outputs  $y'_0, \dots, y'_{n-1}$ .  $\square$

## 4 Locally Simulatable MPC

In this section we introduce the notion of locally simulatable MPC on disconnected graphs.

Towards the definition of locally simulatable MPC, we first recall the standard definition of a functionality to model a function  $f: \mathcal{X}^0 \times \dots \times \mathcal{X}^{\ell-1} \rightarrow \mathcal{Y}^0 \times \dots \times \mathcal{Y}^{\ell-1}$  in Figure 8.

**Functionality  $\mathcal{F}_f$**

The functionality  $\mathcal{F}_f$  is parameterised by the function  $f$  to be computed.

**Input:** The functionality awaits input  $x_i \in \mathcal{X}_i$  from party  $P_i$  (for  $i \in \{0, \dots, \ell - 1\}$ ).

**Computation:** The functionality computes  $(y_0, \dots, y_{\ell-1}) := f(x_0, \dots, x_{\ell-1})$ .

**Output:** The functionality outputs  $y_i$  to party  $P_i$  (for  $i \in \{0, \dots, \ell - 1\}$ ).

Figure 8: Functionality  $\mathcal{F}_f$  for computing  $f: \mathcal{X}^0 \times \dots \times \mathcal{X}^{\ell-1} \rightarrow \mathcal{Y}^0 \times \dots \times \mathcal{Y}^{\ell-1}$ .

We define local simulatability relative to a communication network  $G = (V, E)$ , where  $V = \{0, \dots, \ell - 1\}$ , and where two parties  $P_i$  and  $P_j$  can communicate if and only if they are connected by an edge  $(i, j) \in E$ . In the following we always assume the graph to be connected.

We model the notion of local simulatability, by requiring a simulator to be dividable in simulators  $S_1, \dots, S_\mu$  (one for each connected component of the adversary), where simulator  $S_i$  has to simulate the view of the  $i$ -th component solely based on the inputs and outputs of the parties in this component.

**Real Execution.** Let  $\Pi$  be a protocol executed by parties  $P_0, \dots, P_{\ell-1}$  on  $G$ , i.e., a protocol where each party can only send and receive messages from their neighbors in  $G$ . Then, the view  $\text{View}_i^\Pi(x_0, \dots, x_{\ell-1})$  of party  $P_i$  consists of its input  $x_i$ , its internal randomness  $r_i$  and all messages received by party  $P_j$  with  $(i, j) \in E$ . Let  $\mathcal{A}$  be an adversary corrupting a subset  $I \subset \{0, \dots, \ell - 1\}$  of the players. Then, the view of  $\mathcal{A}$  in the real execution of  $\Pi$  is of the form

$$\text{REAL}_{\mathcal{A}, I}^\Pi(x_0, \dots, x_{\ell-1}) = \left( \Pi(x_0, \dots, x_{\ell-1}), \left\{ \text{View}_i^\Pi(x_0, \dots, x_{\ell-1}) \right\}_{i \in I} \right),$$

where  $\Pi(x_0, \dots, x_{\ell-1})$  denotes the outputs of parties  $P_0, \dots, P_{\ell-1}$  after the execution of  $\Pi$  on input  $(x_0, \dots, x_{\ell-1})$  with randomness  $(r_0, \dots, r_{\ell-1})$ .

**Ideal Execution.** Again, let  $\mathcal{A}$  be an adversary corrupting a subset  $I \subset V$  of the nodes and let  $I_1, \dots, I_\mu$  be a partitioning of  $I$  into *pairwise disconnected components*, i.e. such that

- $I = \bigcup_{j=1}^\mu I_j$
- $I_i, I_j$  are *disconnected* for any  $i \neq j$ , i.e., for each  $u \in I_i$  and  $v \in I_j$  it holds  $(u, v) \notin E$ .

Let  $\text{Sim} = (\text{Sim}_1, \dots, \text{Sim}_\mu)$  be a tuple of algorithms<sup>10</sup>, such that for each  $j \in \{1, \dots, \mu\}$  the following holds:

- $\text{Sim}_j$  is a PPT algorithm,
- $\text{Sim}_j$  obtains an input/ output pair  $(x_i, y_i)$  for all  $i \in I_j$ ,

<sup>10</sup>Note that the distinction into  $\mu$  different simulators instead of  $\mu$  copies of the same simulator is solely for the sake of clarity.

- $\text{Sim}_j$  outputs a simulated view of parties  $\{P_i\}_{i \in I_j}$ .

Then, we define the simulated view of  $\text{Sim}$  in the ideal execution of  $\mathcal{F}_f$  as

$$\text{IDEAL}_{\text{Sim}, I}^f(x_0, \dots, x_{\ell-1}) = \left( f(x_0, \dots, x_{\ell-1}), \left\{ \text{Sim}_j((x_i, y_i)_{i \in I_j}) \right\}_{j \in \mu} \right).$$

**Definition 4.1** (Local Simulation). *Let  $\Pi$  be a protocol on  $G$ . We say that  $\Pi$  emulates  $\mathcal{F}_f$  relative to  $G$  with local simulatability in the static, semi-honest model against  $t$  corruptions if for every PPT adversary  $\mathcal{A}$  corrupting a set  $I \subset \{0, \dots, \ell-1\}$  with  $|I| \leq t$  and for every partitioning of  $I$  into pairwise disconnected components  $I_1, \dots, I_\mu$ , there exists a PPT simulator  $\text{Sim} = (\text{Sim}_1, \dots, \text{Sim}_\mu)$ , such that for all  $x_0, \dots, x_{\ell-1} \in \{0, 1\}^*$  it holds*

$$\left\{ \text{REAL}_{\mathcal{A}, I}^\Pi(x_0, \dots, x_{\ell-1}) \right\} \approx_c \left\{ \text{IDEAL}_{\text{Sim}, I}^f(x_0, \dots, x_{\ell-1}) \right\}$$

#### 4.1 Locally Simulatable Protocols are Execution-Oblivious

In this section we show that the notion of locally simulatability indeed guarantees execution-obliviousness (unless the execution can be derived from the output), as we will require to construct THC. We first formalize what we mean by *execution obliviousness*. In the following we restrict to protocols implementing deterministic functionalities with perfect correctness, i.e. for which  $\Pi(x_0, \dots, x_{\ell-1})$  is well-defined without specifying the random coins. (Note that the requirements on inputs and randomness in the following definition are necessary for preventing a trivial distinguisher.)

**Definition 4.2** (Execution obliviousness.). *Let  $G = (V, E)$  be a graph with  $V = \{0, \dots, \ell-1\}$  and let  $\Pi$  be an  $\ell$ -party protocol on  $G$ . We say  $\Pi$  is execution oblivious on  $G$  tolerating  $t$  corruptions, if for all sets  $I \subseteq \{0, \dots, \ell-1\}$  with  $|I| \leq t$  and for any partitioning of  $I$  into pairwise disconnected components  $I_1, \dots, I_\mu$  the following holds:*

*For all inputs  $(x_0, \dots, x_{\ell-1}), (x_0^{(1)}, \dots, x_{\ell-1}^{(1)}), \dots, (x_0^{(\mu)}, \dots, x_{\ell-1}^{(\mu)}) \in \mathcal{X}_0 \times \dots \times \mathcal{X}_{\ell-1}$  with*

- $x_i^{(j)} = x_i$  for all  $i \in I_j, j \in [\mu]$ , and
- $\Pi(x_0, \dots, x_{\ell-1}) = \Pi(x_0^{(1)}, \dots, x_{\ell-1}^{(1)}) = \dots = \Pi(x_0^{(\mu)}, \dots, x_{\ell-1}^{(\mu)})$ ,

*it holds:*

$$\begin{aligned} & \left( \Pi(x_0, \dots, x_{\ell-1}), \left\{ \text{View}_i^\Pi(x_0, \dots, x_{\ell-1}; r_1, \dots, r_{\ell-1}) \right\}_{i \in I} \right) \\ & \approx_c \left( \Pi(x_0, \dots, x_{\ell-1}), \bigcup_{j=1}^{\mu} \left\{ \text{View}_i^\Pi(x_0^{(j)}, \dots, x_{\ell-1}^{(j)}; r_0^{(j)}, \dots, r_{\ell-1}^{(j)}) \right\}_{i \in I_j} \right), \end{aligned}$$

*where the randomness is taken over the random coins  $r_1, \dots, r_{\ell-1}, \{r_1^{(j)}, \dots, r_{\ell-1}^{(j)}\}_{j \in [\mu]}$ .*

**Lemma 4.3** (Locally Simulatable Protocols are Execution Oblivious). *Let  $G = (V, E)$  be a graph with  $V = \{0, \dots, \ell-1\}$ , let  $\mathcal{F}_f$  be a deterministic  $\ell$ -party functionality and let  $\Pi$  be an  $\ell$ -party protocol on  $G$ . If  $\Pi$  emulates  $\mathcal{F}_f$  relative to  $G$  with local simulatability in the static, semi-honest model against  $t$  corruptions, then  $\Pi$  is execution oblivious on  $G$  tolerating  $t$  corruptions.*

*Proof.* Let  $I \subseteq \{0, \dots, \ell - 1\}$  with  $|I| \leq t$  and let  $I_1, \dots, I_\mu$  be a partitioning of  $I$  into pairwise disconnected components. Let  $(x_0, \dots, x_{\ell-1}), (x_0^{(1)}, \dots, x_{\ell-1}^{(1)}), \dots, (x_0^{(\mu)}, \dots, x_{\ell-1}^{(\mu)}) \in \mathcal{X}_0 \times \dots \times \mathcal{X}_{\ell-1}$  be arbitrary distributions with

- $x_i^{(j)} = x_i$  for all  $i \in I_j, j \in [\mu]$ , and
- $\Pi(x_0, \dots, x_{\ell-1}) = \Pi(x_0^{(1)}, \dots, x_{\ell-1}^{(1)}) = \dots = \Pi(x_0^{(\mu)}, \dots, x_{\ell-1}^{(\mu)})$ ,

Recall that we have to show:

$$\begin{aligned} & \left( \Pi(x_0, \dots, x_{\ell-1}), \left\{ \text{View}_i^\Pi(x_0, \dots, x_{\ell-1}; r_0, \dots, r_{\ell-1}) \right\}_{i \in I} \right) \\ & \approx_c \left( \Pi(x_0, \dots, x_{\ell-1}), \bigcup_{j=1}^{\mu} \left\{ \text{View}_i^\Pi(x_0^{(j)}, \dots, x_{\ell-1}^{(j)}; r_0^{(j)}, \dots, r_{\ell-1}^{(j)}) \right\}_{i \in I_j} \right), \end{aligned}$$

where the randomness is taken over the random coins  $r_1, \dots, r_{\ell-1}, \{r_1^{(j)}, \dots, r_{\ell-1}^{(j)}\}_{j \in [\mu]}$ .

We proceed the proof this via a series of hybrid games.

**Game 0:** The distribution is generated as

$$\left( \Pi(x_0, \dots, x_{\ell-1}), \left\{ \text{View}_i^\Pi(x_0, \dots, x_{\ell-1}; r_1, \dots, r_{\ell-1}) \right\}_{i \in I} \right).$$

This corresponds to the original distribution on the left-hand side.

**Game 1:** The distribution is generated as

$$\left( f(x_0, \dots, x_{\ell-1}), \bigcup_{j=1}^{\mu} \left\{ \text{Sim}_j((x_i, y_i)_{i \in I_j}) \right\} \right),$$

where  $\text{Sim}_1, \dots, \text{Sim}_\mu$  are the simulators due to the local simulatability of  $\Pi$ . By the requirement of local simulatability, the distributions of Game 0 and Game 1 are indistinguishable.

**Game 2.k:** For  $k = 0, \dots, \mu$  the distribution in Game 2.k is generated as

$$\left( f(x_0, \dots, x_{\ell-1}), \bigcup_{j=1}^k \left\{ \text{View}_i^\Pi(x_0^{(j)}, \dots, x_{\ell-1}^{(j)}; r_0^{(j)}, \dots, r_{\ell-1}^{(j)}) \right\}_{i \in I_j} \cup \bigcup_{j=k+1}^{\mu} \left\{ \text{Sim}_j((x_i, y_i)_{i \in I_j}) \right\} \right).$$

To show this step we have to prove indistinguishability of the transition Game 2.(k-1)  $\rightsquigarrow$  2.k:

First, note that we can replace  $f(x_0, \dots, x_{\ell-1})$  by  $f(x_0^{(k)}, \dots, x_{\ell-1}^{(k)})$  and by  $\Pi(x_0^{(k)}, \dots, x_{\ell-1}^{(k)})$  by our assumption of the input. Further, let

$$\text{View}^{\Pi, \leq k} := \bigcup_{j=1}^k \left\{ \text{View}_i^\Pi(x_0^{(j)}, \dots, x_{\ell-1}^{(j)}; r_0^{(j)}, \dots, r_{\ell-1}^{(j)}) \right\}_{i \in I_j}$$

and let

$$\text{Sim}^{\geq k} := \bigcup_{j=k+2}^{\mu} \left\{ \text{Sim}_j((x_i, y_i)_{i \in I_j}) \right\}.$$

we thus have to show that the distribution

$$\left( f(x_0^{(k)}, \dots, x_{\ell-1}^{(k)}), \text{View}^{\Pi, \leq k} \cup \{\text{Sim}_k((x_i, y_i)_{i \in I_k})\} \cup \text{Sim}^{\geq k} \right)$$

is indistinguishable from the distribution

$$\left( \Pi(x_0^{(k)}, \dots, x_{\ell-1}^{(k)}), \text{View}^{\Pi, \leq k} \cup \left\{ \text{View}_i^\Pi(x_0^{(k)}, \dots, x_{\ell-1}^{(k)}; r_0^{(k)}, \dots, r_{\ell-1}^{(k)}) \right\}_{i \in I_k} \cup \text{Sim}^{\geq k} \right).$$

Since  $x_i^{(k)} = x_i$  for all  $i \in I_k$ , this follows from the local simulatability of  $\Pi$ .

**Game 3:** The distribution is generated as

$$\left( \Pi(x_0, \dots, x_{\ell-1}), \bigcup_{j=1}^{\mu} \left\{ \text{View}_i^\Pi(x_0^{(j)}, \dots, x_{\ell-1}^{(j)}; r_0^{(j)}, \dots, r_{\ell-1}^{(j)}) \right\}_{i \in I_j} \right).$$

Note that we get this simulation by replacing  $f(x_0, \dots, x_{\ell-1})$  back with  $\Pi(x_0, \dots, x_{\ell-1})$  in Game 2. $\mu$ . This concludes the proof.  $\square$

## 5 Locally Simulatable Protocol for Directed Paths

In this section, we formally present and prove the locally simulatability of the protocol for computing on a directed path some functionality  $\mathcal{F}$  that is *efficiently invertible from local information* (Definition 3.6). An example of such a functionality is  $\mathcal{F}_{\text{OR}}$ , which we will use to implement THB in the next section. We refer the reader back to Section 2.2 for a detailed overview of the protocol.

### 5.1 The Path Protocol

The directed path protocol  $\Pi_{\text{dir-path}}$  is formally presented in Figure 10. As described in Section 2.2, the protocol works over a directed path  $\text{Path}_\ell = \textcircled{0} \rightarrow \textcircled{1} \cdots \rightarrow \textcircled{\ell-1}$  of length  $\ell = 2^l$ , for some  $l > 0$ . Each party knows its position  $j$  on the path and we refer to each such party as  $\tilde{P}_j$ . The protocol recursively computes the given functionality  $\mathcal{F}$ . Recall that  $\mathcal{F}$  must be *efficiently invertible from local information*, such as  $\mathcal{F}_{\text{OR}}$ , which on input bits  $b_j$  from each party  $\tilde{P}_j$ , outputs  $\bigvee_{j=0}^{\ell-1} b_j$  to every party. When computation of  $\mathcal{F}_{\text{OR}}$  used in our higher-level THB protocol of the next section, the input of party  $\tilde{P}_{j^*}$  corresponding to the broadcaster will be  $b_{j^*} = b$ , the broadcast bit, and for all  $j \neq j^*$ , the input of party  $\tilde{P}_j$  will be  $b_j = 0$ .

As depicted in Figure 9 (reproduced from Section 2.2),  $\Pi_{\text{dir-path}}$  proceeds by recursively emulating a (constant-round, constant-overhead) 2PC that computes  $((y_0 \parallel \dots \parallel y_{\ell/2-1}), (y_{\ell/2} \parallel \dots \parallel y_{\ell-1})) = \mathcal{F}'((x_0 \parallel \dots \parallel x_{\ell/2-1}), (x_{\ell/2} \parallel \dots \parallel x_{\ell-1})) = \mathcal{F}(x_0, \dots, x_{\ell-1})$  for two virtual parties, and then recursively sending the outputs  $y_j$  to the Parties  $P_j$  at the bottom of the recursion tree. Party  $P_{0,0}$  (and similarly for party  $P_{0,1}$ ) of the highest-level 2PC is recursively emulated by parties  $\tilde{P}_0, \dots, \tilde{P}_{\ell/2-1}$  on the path by first computing each message that  $P_{0,0}$  sends in this 2PC via another lower-level 2PC between virtual parties  $P_{1,0}$  and  $P_{1,1}$ . Parties  $P_{1,0}$  and  $P_{1,1}$  combine their inputs and random strings via this 2PC to emulate  $P_{0,0}$ 's input and random tape.  $P_{1,1}$  then receives  $P_{0,0}$ 's next message and sends it to  $P_{1,2}$  (the first party emulating  $P_{0,1}$ ). Continuing in the recursion, both  $P_{1,0}$  and

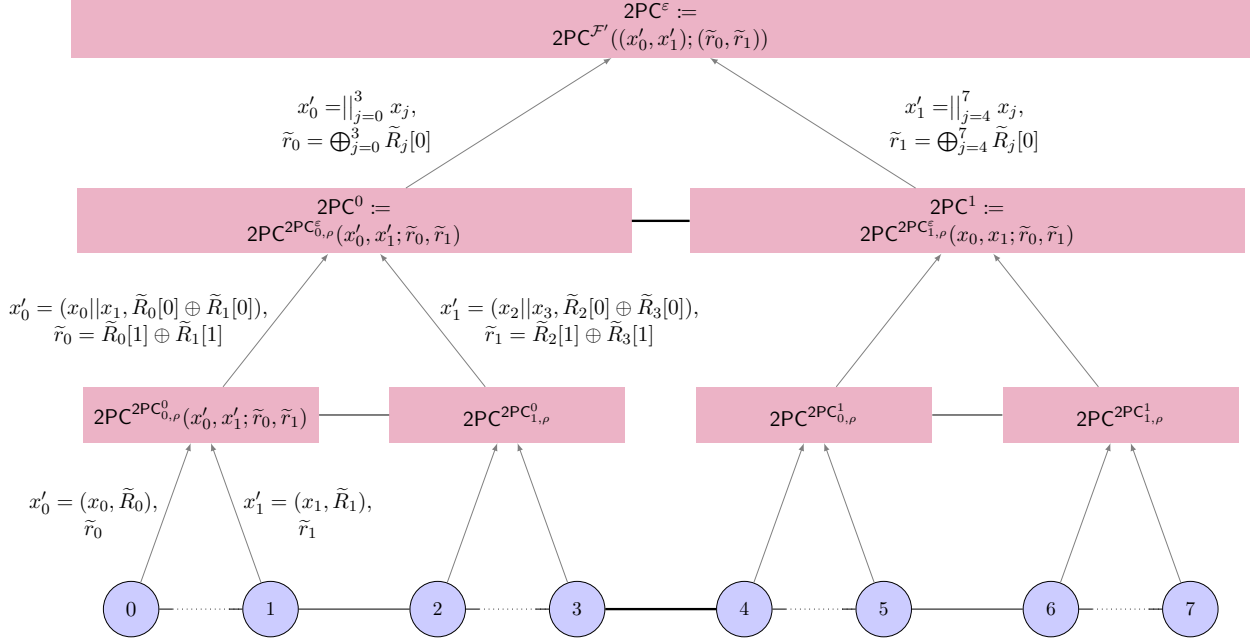


Figure 9: Simplified depiction of directed path protocol  $\Pi_{\text{dir-path}}$  for a path of length  $\ell = 8$ . Each interior node represents a 2PC which gets its inputs and randomness from its two children. This 2PC computes the next message for virtual party  $P_0$  (resp.  $P_1$ ) in the 2PC at the node's parent by combining the inputs of its two children into the input and randomness of  $P_0$  (resp.  $P_1$ ). This next message is passed from this node to its sibling in the protocol via the two *neighboring real* parties at the rightmost (resp. leftmost) and leftmost (resp. rightmost) leaves of the corresponding subtrees (indicated by horizontal lines of matching thickness).

$P_{1,1}$  are then emulated by another 2PC in the same fashion, and so on, until we reach two actual parties on the path.

For each call (either the invocation or recursive calls) to  $\Pi_{\text{dir-path}}$  there are some parameters known to all participants: the current topology being considered (each recursive call works over a connected subgraph of the path); the  $R_{2PC}$  round constant-overhead semi-honest stateless protocol 2PC that is being used for the execution; the recursion depth  $d$ ; the message virtual party  $\sigma \in \{0, 1, \perp\}$  who outputs a message for the 2PC that is being emulated by this instance (if  $\sigma = \perp$ , this means neither party does); output flag  $o \in \{0, 1\}$ , which indicates whether or not the parties produce an output in this execution; and the 2PC functionality  $\mathcal{F}$  that the two virtual parties are computing. For the original invocation call, the path considered is the whole path  $\text{Path}_\ell = \textcircled{0} \rightarrow \textcircled{1} \cdots \rightarrow \textcircled{\ell-1}$ , the recursion depth is  $d = 0$ , message virtual party is  $\sigma = \perp$ , output flag is  $o = 1$ , and the 2PC functionality that will be recursively computed is  $\mathcal{F}'$ ; i.e., on input  $x_0$  from  $P_{0,0}$  (recursively  $\|_{j=0}^{\ell/2-1} x_j$ ) and  $x_1$  from  $P_{0,1}$  (recursively  $\|_{j=\ell/2}^{\ell-1} x_j$ ), output  $\mathcal{F}'(x_0, x_1)$  to  $P_{0,0}$  and  $P_{0,1}$ .

For each call, each party also receives some local input: their position  $j$  on the corresponding subgraph of the path; their input  $x_j$ ; a dictionary of random strings  $\tilde{R}_j$  that they will use for the emulation of high-level 2PC virtual parties; a set of 2PC messages  $M_j$  that they receive from some



higher-level 2PC in which they are assisting the emulation of one of the virtual parties; and their neighbors on the path,  $\tilde{P}_{j-1}$  and  $\tilde{P}_{j+1}$ . For the original invocation call, each party's position is of course  $j$ , their input  $x_j$ , random string dictionary  $\tilde{R}_j[\cdot] = \perp$ , empty message set  $M_j = \emptyset$ , and neighbors  $\tilde{P}_{j-1}$  and  $\tilde{P}_{j+1}$ .

**Efficiency.** Recall that we assume the round complexity of the 2PC protocol is some constant  $R_{2PC}$  and its overhead is  $c \cdot |\mathcal{C}_{\mathcal{F}}| + \text{poly}(\lambda)$  for some constant  $c$ , where  $\mathcal{C}_{\mathcal{F}}$  is the circuit that computes given functionality  $\mathcal{F}$ . Thus, the round complexity of  $\Pi_{\text{dir-path}}$  is  $R[\ell] = 2R_{2PC} \cdot R[\ell/2] + 2 = \Theta(\ell \cdot R_{2PC}^{\log \ell})$ , which is  $O(\ell^2)$ . Furthermore, each real party on the path executes  $R_{2PC}^{\log \ell - 1}$  2PC's. The overhead of the highest-level 2PC is  $c \cdot |\mathcal{C}_{\mathcal{F}}| + \text{poly}(\lambda)$ , the overhead of the 2PC's in the next recursion level are then  $c^2 \cdot |\mathcal{C}_{\mathcal{F}}| + c \cdot \text{poly}(\lambda) + \text{poly}(\lambda)$ , and so on so that the overhead of the 2PC's executed by the real parties is  $O(\ell \cdot (|\mathcal{C}_{\mathcal{F}}| + \text{poly}(\lambda)))$ . Therefore, the total overhead of  $\Pi_{\text{dir-path}}$  is  $O(R_{2PC}^{\log \ell - 1} \cdot \ell \cdot (|\mathcal{C}_{\mathcal{F}}| + \text{poly}(\lambda))) = O(\ell^2 \cdot (|\mathcal{C}_{\mathcal{F}}| + \text{poly}(\lambda)))$ .

### Protocol $\Pi_{\text{dir-path}}$

**Parameters:** A topology  $\text{Path}_{\ell} = \textcircled{0} \rightarrow \textcircled{1} \cdots \rightarrow \textcircled{\ell-1}$  of length  $\ell = 2^l$ , an  $R_{2PC}$ -round constant-overhead semi-honest stateless protocol  $2PC = (2PC_{0,\rho}, 2PC_{1,\rho})_{\rho \in [0, R_{2PC}-1]}$ , recursion depth  $d$ , a message virtual party  $\sigma$ , output flag  $o$ , and 2PC functionality to be (recursively) computed  $\mathcal{F}$ . *Note:* we will use  $\tilde{P}_j$  to denote the party at position  $j$  on the path.

#### The Protocol:

- **Initialisation:** Each party  $\tilde{P}_j$  receives  $(j, x_j, \tilde{R}_j, M_j, \tilde{P}_{j-1}, \tilde{P}_{j+1})$  as input, either from the original invocation, or from a recursive call. Each party  $\tilde{P}_j$  samples  $\tilde{r}_j \xleftarrow{\$} \{0, 1\}^{\lambda}$  and sets  $\tilde{R}_j[d] \leftarrow \tilde{r}_j$ .
- **Base Case ( $\ell = 2$ ):** Parties  $\tilde{P}_0$  and  $\tilde{P}_1$  directly compute the  $R_{2PC}$  round protocol  $2PC^{\mathcal{F}}((x_0, \tilde{R}_0[:d-2], \tilde{R}_0[d-1], M_0), (x_1, \tilde{R}_1[:d-2], \tilde{R}_1[d-1], M_1); \tilde{r}_0, \tilde{r}_1)$ . If  $\sigma \neq \perp$ , then  $\tilde{P}_{\sigma}$  returns the output message to its invoker. Furthermore, if  $o = 1$ , then both parties locally return their outputs  $y_0$  and  $y_1$ , respectively.
- For  $\rho = 0, \dots, R_{2PC} - 2$ :

– For  $k \in [0, 1]$ , rounds  $(2\rho + k) \cdot (R_{\lfloor \frac{\ell}{2} \rfloor} + 1)$  to  $(2\rho + k + 1) \cdot (R_{\lfloor \frac{\ell}{2} \rfloor} + 1) - 1$ :

**Compute virtual Party  $P_k$ 's next 2PC message  $m_{k,\rho}$ .**

1. Parties  $\tilde{P}_{k \cdot \ell/2}, \dots, \tilde{P}_{k \cdot \ell/2 + \ell/2 - 1}$  recursively call  $\Pi_{\text{dir-path}}$  with:

\* **Parameters:**  $\textcircled{k \cdot \ell/2} \rightarrow \textcircled{k \cdot \ell/2 + 1} \cdots \rightarrow \textcircled{k \cdot \ell/2 + \ell/2 - 1}$  of length  $\ell/2 = 2^{l-1}$ , protocol 2PC, recursion depth  $d+1$ , message virtual party  $\sigma' = 1 - k$ , output flag  $o = 0$ , 2PC functionality  $\mathcal{F}_k :=$

•  $P_{k,0}$ 's Input:  $x_{k,0}, \tilde{R}_{k,0}, \tilde{r}_{k,0}, M_{k,0}$  (recursively:

$$\parallel_{j=k \cdot \ell/2}^{k \cdot \ell/2 + \ell/4 - 1} x_j, \left\{ \bigoplus_{j=k \cdot \ell/2}^{k \cdot \ell/2 + \ell/4 - 1} \tilde{R}_j[d'] \right\}_{d' \in [0, d-1]}, \bigoplus_{j=k \cdot \ell/2}^{k \cdot \ell/2 + \ell/4 - 1} \tilde{r}_j, M'_{k \cdot \ell/2};$$

where for  $k = 1$ ,  $M'_{k \cdot \ell/2} = \{m_{0,q}\}_{q < \rho}$ , i.e., messages sent so far by  $P_0$ ;

for  $k = 0$ ,  $M'_{k \cdot \ell/2} = M_{k \cdot \ell/2}$ )

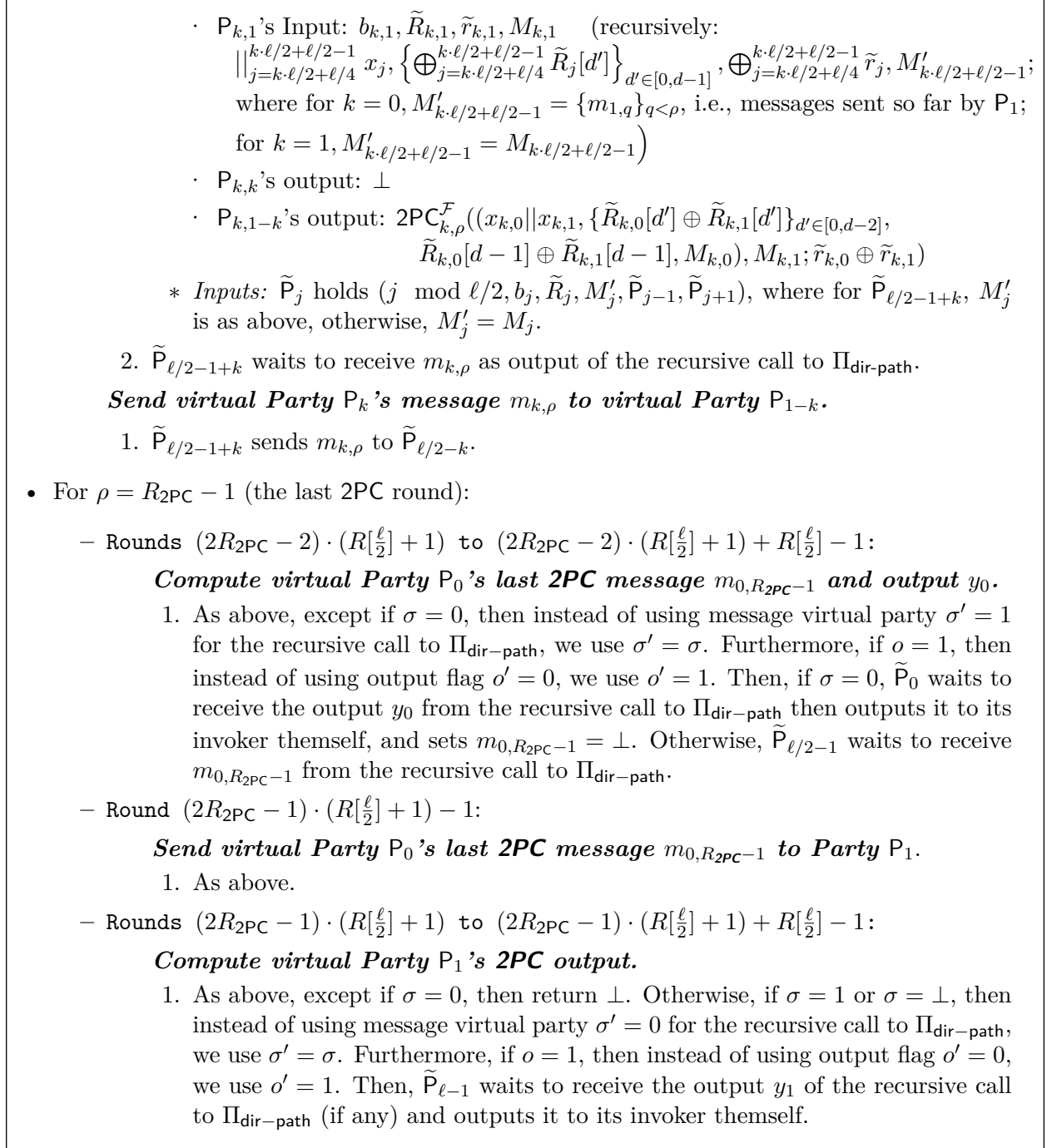


Figure 10: Protocol  $\Pi_{\text{dir-path}}$  which on input  $x_j$  from each party  $\tilde{P}_j$  on a directed path, computes  $\mathcal{F}(x_0, \dots, x_{\ell-1}) = (y_0, \dots, y_{\ell-1})$  and outputs to party  $\tilde{P}_j$  their output  $y_j$ . Note: each party knows their position on the path.

## 5.2 Local Simulatability of the Path Protocol

**Additional Notation.** Before showing our security proof for  $\Pi_{\text{dir-path}}$ , we introduce some more helpful notation.  $\lfloor \cdot \rfloor$  represents rounding to the nearest integer ( $\lfloor x/2 \rfloor = (x+1)/2$  for  $x \equiv 1 \pmod{2}$ ).  $\text{GPo2}(j)$  is the greatest power of 2 that divides  $j$ ; i.e.,  $\text{GPo2}(12) = 4$ , since  $2^2 \mid 12$  but  $2^3 \nmid 12$ .  $\text{nEven}(j)$  is the next closest even number to  $j$ ; i.e.,  $\text{nEven}(3) = \text{nEven}(4) = 4$ .

We index the  $j$ -th node at depth  $d$  in the recursion tree of the protocol with  $(d, j)$ , where  $d$  is counted top down (i.e.,  $d = 0$  corresponds to the root,  $d = \ell$  corresponds to the leaves) and  $j$  is counted from left to right (i.e.,  $j = 0$  is the leftmost node at depth  $d$ ,  $j = 2^d - 1$  is the rightmost node at depth  $d$ ).  $\text{LCAL}(i, j) := \max\{k \geq 0 : \lfloor i/2^k \rfloor = \lfloor j/2^k \rfloor\}$  is the number of levels up the recursion tree where the respective paths of leaves  $i$  and  $j$  to the root meet.

$\mathcal{F}_{d,j,k}$  is the  $k$ -th functionality computed at node  $j$  at depth  $d$ , for  $k \in [0, R_{2\text{PC}}^d]$ .  $\text{parID} := (d-1, \lfloor j/2 \rfloor, \lfloor k/R_{2\text{PC}} \rfloor)$  is the identifier of the functionality at the parent of node  $j$  at depth  $d$  in the recursion tree for which  $\mathcal{F}_{d,j,k}$  is completely emulating a virtual party.  $\mathcal{F}_{d,j,k}^{\text{par}} := \mathcal{F}_{\text{parID}}$  is the corresponding functionality,  $i_{\text{par}} := j \bmod 2$  is the corresponding virtual party, and  $\rho_{d,j,k}^{\text{par}} := k \bmod R_{2\text{PC}}$  is the round of  $\mathcal{F}_{d,j,k}^{\text{par}}$  which  $\mathcal{F}_{d,j,k}$  emulates.  $\text{ancID} := (d - \log(\text{GPo2}(\text{nEven}(j))) - 1, \lfloor j/(2 \cdot \text{GPo2}(\text{nEven}(j))) \rfloor, \lfloor k/R_{2\text{PC}}^{\log(\text{GPo2}(\text{nEven}(j)))+1} \rfloor)$  is the identifier of the higher-level functionality at the ancestor of node  $j$  at depth  $d$  in the recursion tree whose 2PC execution's messages  $\mathcal{F}_{d,j,k}$  also receives as input and for which  $\mathcal{F}_{d,j,k}$  assists in the emulation of a virtual party (i.e., it does not completely emulate the virtual party).  $\mathcal{F}_{d,j,k}^{\text{anc}} := \mathcal{F}_{\text{ancID}}$  is the corresponding functionality,  $i_{\text{anc}} := \lfloor j/(2 \cdot \text{GPo2}(\text{nEven}(j))) - \lfloor j/(2 \cdot \text{GPo2}(\text{nEven}(j))) \rfloor \rfloor$  is the corresponding virtual party, and  $\rho_{d,j,k}^{\text{anc}} := \lfloor k/R_{2\text{PC}}^{\log(\text{GPo2}(\text{nEven}(j)))} \rfloor \bmod R_{2\text{PC}}$  is the round of  $\mathcal{F}_{d,j,k}^{\text{anc}}$  whose emulation  $\mathcal{F}_{d,j,k}$  assists.

Party  $i_{\text{from-par}} \in \{0, 1\}$  for  $\mathcal{F}_{d,j,k}$  is the virtual party in this 2PC which receives the messages from the emulation of  $2\text{PC}_{i_{\text{par}}}^{\mathcal{F}_{d,j,k}^{\text{par}}}$ .  $i_{\text{from-an}} = 1 - i_{\text{from-par}} \in \{0, 1\}$  for  $\mathcal{F}_{d,j,k}$  is the virtual party in this 2PC which does not receive any messages (if  $j \in \{0, 2^d - 1\}$ ), or which receives messages from the emulation of  $2\text{PC}_{i_{\text{anc}}}^{\mathcal{F}_{d,j,k}^{\text{anc}}}$ .

Finally, given some recursion depth  $d$ , node  $j$ , and virtual party  $P_i$  of the 2PC functionality  $\mathcal{F}_{d,j,k}$ , based on the subset  $J \subseteq [0, \ell - 1]$  of the real parties of which  $P_i$  consists, and their corresponding inputs and outputs of the functionality  $\mathcal{F}$  that the whole protocol computes,  $\{(x_j, y_j)\}_{j \in J} \subseteq \{(x_0, y_0), \dots, (x_{\ell-1}, y_{\ell-1})\}$ , let the inverted inputs of the other real parties be  $\{x'_j\}_{j \in [0, \ell-1] \setminus J} = \mathcal{F}^{-1}(\{(x_j, y_j)\}_{j \in J})$  (recall that  $\mathcal{F}$  is efficiently invertible from local information). We recursively define  $M_{\text{fake}}^{d,j,k,i}$  as

$$M_{\text{fake}}^{d,j,k,i} := \begin{cases} \emptyset & j \in \{0, 2^d - 1\} \\ 2\text{PC}_{1-i_{\text{par}}, < \rho_{d,j,k}^{\text{par}}}^{\mathcal{F}_{d,j,k}^{\text{par}}} \left( \left( x'_{i_{\text{par}}}, \tilde{R}_{i_{\text{par}}}, \tilde{r}_{i_{\text{par}}}, M_{\text{fake}}^{\text{parID}, i_{\text{par}}} \right), \right. \\ \quad \left. \left( x'_{1-i_{\text{par}}}, \tilde{R}_{1-i_{\text{par}}}, \tilde{r}_{1-i_{\text{par}}}, M_{\text{fake}}^{\text{parID}, 1-i_{\text{par}}} \right) \right) & i = i_{\text{from-par}} \\ 2\text{PC}_{1-i_{\text{anc}}, < \rho_{d,j,k}^{\text{anc}}}^{\mathcal{F}_{d,j,k}^{\text{anc}}} \left( \left( x'_{i_{\text{anc}}}, \tilde{R}_{i_{\text{anc}}}, \tilde{r}_{i_{\text{anc}}}, M_{\text{fake}}^{\text{ancID}, i_{\text{anc}}} \right), \right. \\ \quad \left. \left( x'_{1-i_{\text{anc}}}, \tilde{R}_{1-i_{\text{anc}}}, \tilde{r}_{1-i_{\text{anc}}}, M_{\text{fake}}^{\text{ancID}, 1-i_{\text{anc}}} \right) \right) & i = i_{\text{from-an}}, \end{cases}$$

for  $x'_{i_{\text{par}}}, x'_{1-i_{\text{par}}}, x'_{i_{\text{anc}}}, x'_{1-i_{\text{anc}}}$  based on real party inputs  $\{x_j\}_{j \in J}$  and real party inverted inputs  $\{x'_j\}_{j \in [0, \ell-1] \setminus J}$ , and randomly sampled  $\tilde{R}_{i_{\text{par}}}, \tilde{R}_{i_{\text{anc}}}, \tilde{R}_{1-i_{\text{par}}}, \tilde{R}_{1-i_{\text{anc}}}, \tilde{r}_{i_{\text{par}}}, \tilde{r}_{i_{\text{anc}}}, \tilde{r}_{1-i_{\text{par}}}, \tilde{r}_{1-i_{\text{anc}}}$ .

**Simulation Helper Lemma.** We first introduce a helpful lemma which will aid us in proving the local simulatability of  $\Pi_{\text{dir-path}}$ .

**Lemma 5.1** (Simulation Helper Lemma). *For every recursion depth  $d \in [0, l-1]$ , every 2PC node  $j \in [0, 2^d - 1]$  at depth  $d$ , every 2PC execution  $k \in [0, R_{2\text{PC}}^d]$  at this node, and each 2PC party  $i \in \{0, 1\}$  of this 2PC, let  $J \subseteq [0, \ell-1]$  be the real parties of which virtual party  $P_i$  consists,  $\{(x_j, y_j)\}_{j \in J}$  their inputs and outputs of the functionality  $\mathcal{F}$  that  $\Pi_{\text{dir-path}}$  computes, and  $\{x'_j\}_{j \in [0, \ell-1] \setminus J}$  be inputs of all other real parties that are input-consistent with  $\{(x_j, y_j)\}_{j \in J}$ . Then:*

$$\mathcal{D}_{d,j,k,i}^0 := \text{VIEW}_i^{2\text{PC}^{\mathcal{F}_{d,j,k}}}((x_i, \tilde{R}_i, \tilde{r}_i, M_i), (x_{1-i}, \tilde{R}_{1-i}, \tilde{r}_{1-i}, M_{1-i})) \approx_c$$

$$\mathcal{D}_{d,j,k,i}^1 := \text{VIEW}_i^{2\text{PC}^{\mathcal{F}_{d,j,k}}}((x_i, \tilde{R}_i, \tilde{r}_i, M_i), (x'_{1-i}, \tilde{R}_{1-i}, \tilde{r}_{1-i}, M_{\text{fake}}^{d,j,k,1-i})),$$

for  $x'_{1-i}, M_{\text{fake}}^{d,j,k,1-i}$  based on input-consistent  $\{x'_j\}_{j \in [0, \ell-1] \setminus J}$ .

*Proof.* We will proceed by induction on the depth  $d$  of recursion.

- **Base Case** –  $d = 0$ : Here we want to prove that for  $i \in \{0, 1\}$ ,  $\mathcal{D}_{0,0,0,i}^0 := \text{VIEW}_i^{\mathcal{F}'}((x_i, \tilde{R}_i, \perp, \emptyset), (x_{1-i}, \tilde{R}_{1-i}, \perp, \emptyset)) \approx_c \mathcal{D}_{0,0,0,i}^1 := \text{VIEW}_i^{\mathcal{F}'}((x_i, \tilde{R}_i, \perp, \emptyset), (x'_{1-i}, \tilde{R}_{1-i}, \perp, \emptyset))$ , where  $\tilde{R}_i[\cdot] = \tilde{R}_{1-i}[\cdot] = \perp$ . This simply follows by 2PC security and the fact that  $x'_{1-i}$  is input-consistent with  $(x_i, y_i)$ . We have that  $\text{VIEW}_i^{\mathcal{F}'}((x_i, \tilde{R}_i, \perp, \emptyset), (x_{1-i}, \tilde{R}_{1-i}, \perp, \emptyset)) \approx_c \text{Sim}_i((x_i, \tilde{R}_i, \perp, \emptyset), y_i) \approx_c \text{VIEW}_i^{\mathcal{F}'}((x_i, \tilde{R}_i, \perp, \emptyset), (x'_{1-i}, \tilde{R}_{1-i}, \perp, \emptyset))$ .
- **Inductive Hypothesis:** For all recursion depths  $d' \in [0, d]$ , every 2PC node  $j \in [0, 2^{d'} - 1]$  at depth  $d'$ , every 2PC execution  $k \in [0, R_{2\text{PC}}^{d'}]$  at this node, and each 2PC party  $i \in \{0, 1\}$  of this 2PC, let  $\{(x_j, y_j)\}_{j \in J}$  be the input and outputs of the real parties of the virtual party  $P_i$  and  $\{x'_j\}_{j \in [0, \ell-1] \setminus J}$  be inputs of all other real parties input-consistent with  $\{(x_j, y_j)\}$ . Then  $\mathcal{D}_{d',j,k,i}^0 \approx_c \mathcal{D}_{d',j,k,i}^1$ .
- **Inductive Step:** Given the inductive hypothesis for depth  $d$ , we want to show that for depth  $d+1$ , for every 2PC node  $j \in [0, 2^{d+1} - 1]$ , every 2PC execution  $k \in [0, R_{2\text{PC}}^{d+1}]$  at this node, and each 2PC party  $i \in \{0, 1\}$  of this 2PC, if  $\{(x_j, y_j)\}_{j \in J}$  are the input and outputs of the real parties of the virtual party  $P_i$  and  $\{x'_j\}_{j \in [0, \ell-1] \setminus J}$  are inputs of all other real parties input-consistent with  $\{(x_j, y_j)\}$ , then  $\mathcal{D}_{d+1,j,k,i}^0 \approx_c \mathcal{D}_{d+1,j,k,i}^1$ .

For given  $j, k$ , let's assume  $i = i_{\text{from-par}}$ . We know from the inductive hypothesis that

$$2\text{PC}_{i_{\text{par}}, \rho_{d,j,k}^{\text{par}}}^{\mathcal{F}_{d,j,k}^{\text{par}}}((x_{i_{\text{par}}}, \tilde{R}_{i_{\text{par}}}, \tilde{r}_{i_{\text{par}}}, M_{i_{\text{par}}}), M_{i_{\text{from-par}}}) \approx_c 2\text{PC}_{i_{\text{par}}, \rho_{d,j,k}^{\text{par}}}^{\mathcal{F}_{d,j,k}^{\text{par}}}((x'_{i_{\text{par}}}, \tilde{R}_{i_{\text{par}}}, \tilde{r}_{i_{\text{par}}}, M_{\text{fake}}^{\text{parID}, i_{\text{par}}}), M_{i_{\text{from-par}}}),$$

since all of  $x'_{i_{\text{par}}}$  and  $x'_{1-i_{\text{par}}}$  are input-consistent with  $\{(x_j, y_j)\}$  (and symmetrically by Lemma 3.7) and so these two messages correspond to the same part of indistinguishable distributions  $\mathcal{D}_{\text{parID}, 1-i_{\text{par}}}^0$  and  $\mathcal{D}_{\text{parID}, 1-i_{\text{par}}}^1$ , respectively. The inductive step thus follows by 2PC security and the above argument:

$$\text{VIEW}_{i_{\text{from-par}}}^{2\text{PC}^{\mathcal{F}_{d,j,k,i}}}((x_{i_{\text{from-par}}}, \tilde{R}_{i_{\text{from-par}}}, \tilde{r}_{i_{\text{from-par}}}, M_{i_{\text{from-par}}}), (x_{1-i_{\text{from-par}}}, \tilde{R}_{1-i_{\text{from-par}}}, \tilde{r}_{1-i_{\text{from-par}}}, M_{1-i_{\text{from-par}}})) \approx_c$$

$$\begin{aligned} \text{Sim}_{i_{\text{from-par}}}((x_{i_{\text{from-par}}}, \tilde{R}_{i_{\text{from-par}}}, \tilde{r}_{i_{\text{from-par}}}, M_{i_{\text{from-par}}}), 2\text{PC}_{i_{\text{par}}, \rho_{d,j,k}^{\text{par}}}^{\mathcal{F}_{d,j,k}^{\text{par}}}((x_{i_{\text{par}}}, \tilde{R}_{i_{\text{par}}}, \tilde{r}_{i_{\text{par}}}, M_{i_{\text{par}}}), M_{i_{\text{from-par}}})) \approx_c \\ \text{Sim}_{i_{\text{from-par}}}((x_{i_{\text{from-par}}}, \tilde{R}_{i_{\text{from-par}}}, \tilde{r}_{i_{\text{from-par}}}, M_{i_{\text{from-par}}}), 2\text{PC}_{i_{\text{par}}, \rho_{d,j,k}^{\text{par}}}^{\mathcal{F}_{d,j,k}^{\text{par}}}((x'_{i_{\text{par}}}, \tilde{R}_{i_{\text{par}}}, \tilde{r}_{i_{\text{par}}}, M_{\text{fake}}^{\text{parID}, i_{\text{par}}}), M_{i_{\text{from-par}}})) \approx_c \\ \text{VIEW}_{i_{\text{from-par}}}^{2\text{PC}_{d,j,k,i}^{\mathcal{F}_{d,j,k,i}}}((x_{i_{\text{from-par}}}, \tilde{R}_{i_{\text{from-par}}}, \tilde{r}_{i_{\text{from-par}}}, M_{i_{\text{from-par}}}), (x'_{1-i_{\text{from-par}}}, \tilde{R}_{1-i_{\text{from-par}}}, \tilde{r}_{1-i_{\text{from-par}}}, M_{\text{fake}}^{d,j,k,1-i_{\text{from-par}}})) \end{aligned}$$

The analysis follows similarly for  $i_{\text{from-an}}$ .

□

**Local Simulatability of  $\Pi_{\text{dir-path}}$ .** Using Lemma 5.1 and simulator  $\text{Sim}_{\text{dir-path}}$  of Figure 11, we prove in Theorem 5.2 that  $\Pi_{\text{dir-path}}$  is locally simulatable on the path.

### Simulator $\text{Sim}_{\text{dir-path}}$

**Parameters:** A path topology  $\text{Path}_\ell = \textcircled{0} \rightarrow \textcircled{1} \cdots \rightarrow \textcircled{\ell-1}$  of length  $\ell = 2^l$ , an  $R_{2\text{PC}}$ -round semi-honest protocol  $2\text{PC} = (2\text{PC}_{0,i}, 2\text{PC}_{1,i})_{i \in [0, R_{2\text{PC}}-1]}$ , a block of corrupted parties  $[i, j]$  such that  $0 \leq i \leq j \leq \ell - 1$ , and their input, output pairs  $(x_i, y_i), \dots, (x_j, y_j)$ .

**Simulator:** First compute  $\{x'_j\}_{j \in [0, \ell-1] \setminus [i, j]} = \mathcal{F}^{-1}(\{(x_m, y_m)\}_{m \in [i, j]})$ .

- **Interior Corrupted Parties:** For every  $i < k < j$ , proceed as in the real world.
- **Corrupted Endpoint Parties Cases:**
  1.  $\text{LCAL}(i, j) = l$ ; i.e., the least-common ancestor of leaves  $i$  and  $j$  in the recursion tree is the root.
    - *Simulation for  $i$ :* If  $i \bmod 2 = 1$ , then for every functionality  $\mathcal{F}_{1-1, \lfloor i/2 \rfloor, k}$  that  $i$  computes with its neighbor  $i - 1$ , simulate  $i$ 's view using  $\text{Sim}((x_i, \tilde{R}_i, \tilde{r}_i, M_i), \text{OUTPUT}_1^{2\text{PC}_{1-1, \lfloor i/2 \rfloor, k}^{\mathcal{F}_{1-1, \lfloor i/2 \rfloor, k}}}((x_i, \tilde{R}_i, \tilde{r}_i, M_i), (x'_{1-i}, \tilde{R}_{1-i}, \tilde{r}_{1-i}, M_{\text{fake}}^{l-1, \lfloor i/2 \rfloor, k, 1-i})))$ , for freshly sampled randomness  $\tilde{R}_{1-i}, \tilde{r}_{1-i}$ . Otherwise, if  $i = i_{\text{from-par}}$ , then for every functionality  $\mathcal{F}_{1-1, \lfloor i/2 \rfloor, k}^{\text{par}}$  that  $i_{\text{par}}$  computes with its neighbor  $1 - i_{\text{par}}$ , simulate  $i_{\text{par}}$ 's view using  $\text{Sim}((x_{i_{\text{par}}}, \tilde{R}_{i_{\text{par}}}, \tilde{r}_{i_{\text{par}}}, M_{i_{\text{par}}}), \text{OUTPUT}_{i_{\text{par}}}^{2\text{PC}_{1-1, \lfloor i/2 \rfloor, k}^{\text{par}}}((x_{i_{\text{par}}}, \tilde{R}_{i_{\text{par}}}, \tilde{r}_{i_{\text{par}}}, M_{i_{\text{par}}}), (x'_{1-i_{\text{par}}}, \tilde{R}_{1-i_{\text{par}}}, \tilde{r}_{1-i_{\text{par}}}, M_{\text{fake}}^{\text{parID}, 1-i_{\text{par}}}))$ ), for freshly sampled randomness  $\tilde{R}_{1-i_{\text{par}}}, \tilde{r}_{1-i_{\text{par}}}$ , and similarly for if  $i = i_{\text{from-an}}$ . Note that since  $\text{LCAL}(i, j) = l$ , we can recover all of  $i_{\text{par}}$ 's (resp.  $i_{\text{anc}}$ 's) input, as well as its output.
    - *Simulation for  $j$ :* Analogous to above.
  2.  $\text{LCAL}(i, j) < l$ ; i.e., the least-common ancestor of leaves  $i$  and  $j$  in the recursion tree is not the root. Assume w.l.o.g. that  $i = \arg \max_{m \in \{i, j+1\}} \text{GPO2}(m)$  ( $\text{Sim}_{\text{dir-path}}$  proceeds similarly for the other case). If  $j \geq i + \text{GPO2}(i) - 1$  then proceed as above. Otherwise:
    - *Simulation for  $i$ :* If  $i = i_{\text{from-an}}$ , then for every functionality  $\mathcal{F}_{1-1, \lfloor i/2 \rfloor, k}^{\text{anc}}$  that  $i_{\text{anc}}$  computes with its neighbor  $1 - i_{\text{anc}}$ , simulate  $i_{\text{anc}}$ 's view using  $\text{Sim}((x'_{i_{\text{anc}}}, \tilde{R}_{i_{\text{anc}}}, \tilde{r}_{i_{\text{anc}}}, M_{\text{fake}}^{\text{ancID}, i_{\text{anc}}}), \text{OUTPUT}_{i_{\text{anc}}}^{2\text{PC}_{1-1, \lfloor i/2 \rfloor, k}^{\text{anc}}}((x'_{i_{\text{anc}}}, \tilde{R}_{i_{\text{anc}}}, \tilde{r}_{i_{\text{anc}}}, M_{\text{fake}}^{\text{ancID}, i_{\text{anc}}}))$ ),

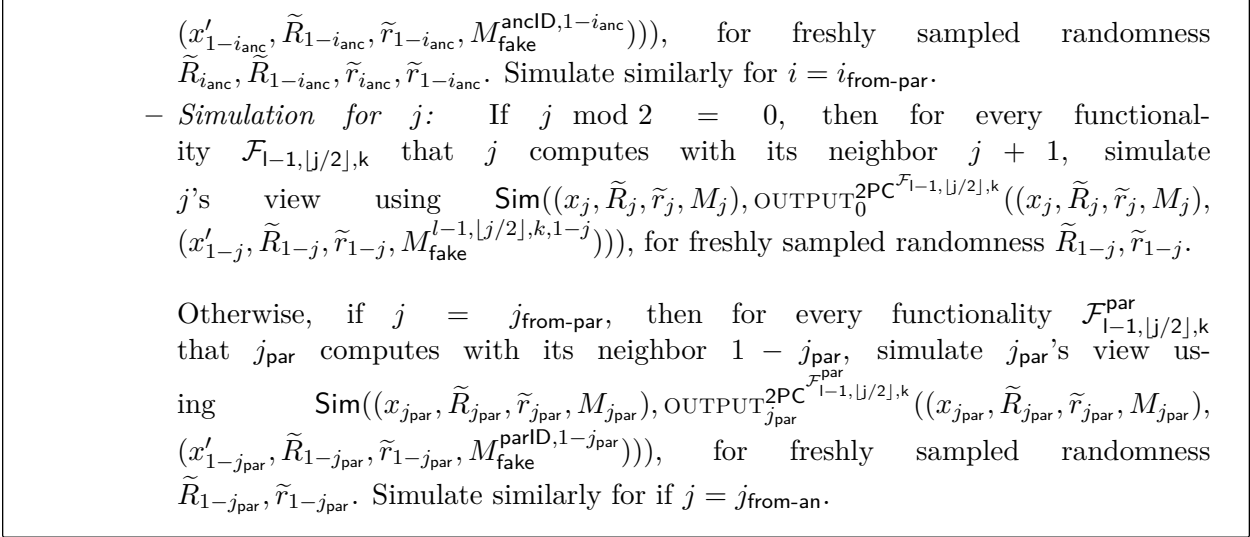


Figure 11: Simulator  $\text{Sim}_{\text{dir-path}}$  for directed paths protocol  $\Pi_{\text{dir-path}}$ .

**Theorem 5.2** (Local Simulatability of  $\Pi_{\text{dir-path}}$ ). *For any functionality  $\mathcal{F}$  that is efficiently invertible from local information,  $\Pi_{\text{dir-path}}$  emulates  $\mathcal{F}$  relative to  $\text{Path}_\ell$  with local simulatability in the static, semi-honest model against  $\ell$  corruptions.*

*Proof.* For each block of corrupted parties  $I_k = [i_k, j_k]$  such that  $0 \leq i_k \leq j_k \leq \ell - 1$ , we set  $\text{Sim}_k = \text{Sim}_{\text{dir-path}}$  of Figure 11. To show that the real and simulated views are indistinguishable, we one-by-one, according to an order described below, replace the view of some endpoint of some block of corrupted parties  $I_k = [i_k, j_k]$  with the view that is generated by  $\text{Sim}_{\text{dir-path}}$  for them. The endpoints are ordered by the depth of the 2PC functionality for which  $\text{Sim}_{\text{dir-path}}$  invokes the corresponding 2PC simulator for them, with lower depths first. Formally, this corresponds to  $l - \text{GPo2}(i_k)$  and  $l - \text{GPo2}(j_k + 1)$  for each such  $i_k$  and  $j_k$  (where larger values correspond to *lower* depths that come first in the order). I.e., we work our way up from the bottom to the top of the recursion tree. For example, those  $i_k$  and  $j_k$  such that  $i_k \bmod 2 = 1$  and  $j_k \bmod 2 = 0$  come first, as then the 2PC simulator invoked for them is for depth  $l - 1$  functionalities. Local simulatability is easily achieved by following this order, since for a corrupted party participating in a lower-level 2PC, any messages input or output to the other party in the 2PC, which are part of the view of a corrupted virtual party in a higher-level 2PC, should look independent of their view. Thus when the lower level corrupted party's real view is replaced by a simulated view, the adversary as a whole, even with the higher-level corrupted virtual party's still real view, does not notice any change. For each endpoint in this order, we proceed by the cases identified in Figure 11:

1.  $i_k$  is in some  $I_k = [i_k, j_k]$  such that  $\text{LCAL}(i_k, j_k) = l$ . In this case, let  $\text{HYB}_0$  be  $i$ 's current view and  $\text{HYB}_{0,1}$  be the world in which we replace  $1 - i$ 's (or  $1 - i_{\text{par}}$ 's, or  $1 - i_{\text{anc}}$ 's) input with  $x'_{1-i}$ , and their input message set with  $M_{\text{fake}}^{l-1, \lfloor i/2 \rfloor, k, 1-i}$  (even though this may not be true in the real world).  $\text{HYB}_0 \approx_c \text{HYB}_{0,1}$  then directly follows from Lemma 5.1. Now let  $\text{HYB}_1$  be the world in which we replace  $i$ 's (or  $i_{\text{par}}$ 's, or  $i_{\text{anc}}$ 's) view with the Simulation for  $i$  as detailed in Figure 11. Then  $\text{HYB}_{0,1} \approx_c \text{HYB}_1$  simply follows by a standard hybrid argument reducing to the security of protocol 2PC, for each 2PC execution that  $i_k$  participates in. A similar argument follows for  $j_k$ .



2.  $i_k$  (resp.  $j_k$ ) is in some  $I_k = [i_k, j_k]$  such that  $\text{LCAL}(i_k, j_k) < l$ . Assume w.l.o.g. that  $i_k = \arg \max_{m \in \{i_k, j_k+1\}} \text{GPo2}(m)$  (the proof proceeds similarly for the other case). If  $j_k \geq i_k + \text{GPo2}(i_k) - 1$  then proceed as above. Otherwise, we proceed as follows:

If we are currently doing a hybrid argument for some  $j = j_k$ 's view, then let hybrid  $\text{HYB}_0$  be  $j$ 's current view and  $\text{HYB}_{0,1}$  be the world in which we replace  $1-j$ 's (or  $1-j_{\text{par}}$ 's, or  $1-j_{\text{anc}}$ 's) input with  $x'_{1-j}$ , and their input message set with  $M_{\text{fake}}^{l-1, \lfloor j/2 \rfloor, k, 1-j}$  (even though this may not be true in the real-world).  $\text{HYB}_0 \approx_c \text{HYB}_{0,1}$  then directly follows from Lemma 5.1. Now let  $\text{HYB}_1$  be the world in which we replace  $j$ 's (or  $j_{\text{par}}$ 's, or  $j_{\text{anc}}$ 's) view with the Simulation for  $j$  as detailed in Figure 11. Then again, we easily see that  $\text{HYB}_{0,1} \approx_c \text{HYB}_1$  via standard reductions to 2PC security.

Otherwise, if we are currently doing a hybrid argument for some  $i = i_k$ 's view, first note that the corresponding  $j_k$ 's view will have already been replaced as above, because of the ordering of our hybrids. Now, assume that  $i = i_{\text{from-anc}}$  (the other case follows similarly) and let  $\text{HYB}_0$  be  $i$ 's current view.  $\text{HYB}_{0,1}$  replaces  $i_{\text{anc}}$ 's view with a generated view (with freshly sampled randomness) in which  $i_{\text{anc}}$  and  $1-i_{\text{anc}}$  have inputs  $x'_{i_{\text{anc}}}$ ,  $x'_{1-i_{\text{anc}}}$ , respectively, and input messages  $M_{\text{fake}}^{\text{anclD}, i_{\text{anc}}}$ ,  $M_{\text{fake}}^{\text{anclD}, 1-i_{\text{anc}}}$ , respectively (even though this may not be true in the real-world). First note again that  $j_k$ 's corresponding view will have already been replaced as above, so that in  $\text{HYB}_0$  we already have a simulation consistent with  $i_{\text{anc}}$ 's input being  $x'_{i_{\text{anc}}}$  and input message set being  $M_{\text{fake}}^{\text{anclD}, i_{\text{anc}}}$ . Now, it directly follows from Lemma 5.1 that replacing  $1-i_{\text{anc}}$ 's input and message set similarly is computationally indistinguishable. Therefore,  $\text{HYB}_0 \approx_c \text{HYB}_{0,1}$ . Now, in hybrid  $\text{HYB}_1$ , we finally replace  $i_{\text{anc}}$ 's view with the Simulation for  $i_{\text{anc}}$  as detailed in Figure 11. Again, it can easily be seen that  $\text{HYB}_{0,1} \approx_c \text{HYB}_1$  via standard reductions to 2PC security, and since the uncorrupted parties in  $i_{\text{anc}}$  and  $1-i_{\text{anc}}$  use independently sampled randomness.  $\square$

## 6 Extension to All Graphs

In this section we show how to use our protocol for local simulation on paths to achieve topology-hiding computation on all graphs, building on the techniques of [ALM17].

### 6.1 From Local Simulation on Paths to Topology-Hiding Computation on All Graphs

Let  $B$  be an upper bound of the size of the number of parties in the graph. Recall that by [ALM17], if we set  $\ell \geq 8\kappa \cdot B^3$ , then a correlated random walk, as described in the introduction and more formally in Protocol  $\Pi_{\text{all-graphs}}$  (Fig.  $\Pi_{\text{all-graphs}}$ ) below, covers the graph except with probability  $2^{-\kappa}$ . Since our protocol for computing OR on a path with local simulatability requires the underlying path to be a power of 2, we set  $\ell = 2^{\lceil 8\kappa \cdot B^3 \rceil}$ . In the following we first present the functionality  $\mathcal{F}_{\text{OR}}$  (Fig. 12) (recall that this is equivalent to broadcast), and then our protocol  $\Pi_{\text{all-graphs}}$  (Fig. 13) implementing  $\mathcal{F}_{\text{OR}}$  on all graphs.

We outline our protocol in Figure 13. We restrict in our description to the neighborhood model  $\text{KT}_1$ , but it is straightforward to generalize to  $\text{KT}_0$ , since one can replace a permutation on the real neighbourhood simply by a permutation of the pseudonymised neighbourhood.

**Functionality  $\mathcal{F}_{\text{OR}}$**

**Initialisation:** The functionality receives the communication graph  $G$  from the wrapper  $\mathcal{W}_{\text{graph-info}}$ .

**Input:** Record the input bit  $x_i$  of each  $P_i$ .

**Output:** For each connected component  $X \subseteq V$  in  $G$ , compute  $b_X \leftarrow \bigvee_{P_i \in X} x_i$  (the boolean OR of the inputs of all parties in the component), and send the output  $X$  to every party in  $X$ .

Figure 12: The OR functionality

**Protocol  $\Pi_{\text{all-graphs}}$**

**Public Parameters:**  $B$  is an upper bound on the number of parties in the graph, and  $\kappa$  is a correctness parameter. The common length of the random walks is set to  $\ell := 2^{\lceil \log(8\kappa \cdot B^3) \rceil}$ .

**Requires:**  $\Pi_{\text{dir-path}} = (\text{Init}, \text{next}^{\text{dir-path}}, \text{RetrieveOutput})$  is an  $R_{\text{dir-path}}$ -round locally simulatable protocol for securely computing  $(x_0, \dots, x_{\ell-1}) \mapsto \bigvee_{i=0}^{\ell-1} x_i$  on the directed path  $\textcircled{0} \rightarrow \textcircled{1} \dots \rightarrow \textcircled{\ell-1}$ , of length  $\ell$ .

- **Init:** On input a position  $i \in \{0, \dots, \ell-1\}$  on the path, the security parameter  $1^\lambda$ , and an input  $x_i$ , **Init** outputs the initial state  $\text{st}_i^{(0)}$  of the party in position  $\textcircled{i}$ .
- **next<sup>dir-path</sup>:** On input a position  $i \in [\ell]$ , a round number  $r$ , the message  $m_{i,-}^{(r-1)}$  sent from  $\textcircled{i-1}$  to  $\textcircled{i}$  in round  $r-1$  (or  $\perp$  if  $i=0$  or  $r=1$ ), the message  $m_{i,+}^{(r-1)}$  sent from  $\textcircled{i+1}$  to  $\textcircled{i}$  in round  $r-1$  (or  $\perp$  if  $i=\ell-1$  or  $r=1$ ), and party  $\textcircled{i}$ 's state  $\text{st}_i^{(r-1)}$  at round  $r-1$ , **next<sup>dir-path</sup>** outputs messages  $m_{i-1,+}^{(r)}$  (or  $\perp$  if  $i=0$ ),  $m_{i+1,-}^{(r)}$  (or  $\perp$  if  $i=\ell-1$ ) and  $\textcircled{i}$ 's updated state at round  $\text{st}_i^{(r)}$ .
- **RetrieveOutput:** On input  $\textcircled{\ell}$ 's final state  $\text{st}_\ell^{(R_{\text{dir-path}})}$ , **RetrieveOutput** produces its output.

**Hybrid Model:** The protocol is defined in the  $\mathcal{F}_{\text{graph}}^{\mathcal{G}}$ -hybrid model.

**The Parties:** Each party  $P_u$  is uniquely identified by some  $u$  in  $[B]$ .

**The Protocol:**

- **Initialisation:** Upon receiving the input  $m_u$  from the environment, each party  $P_u$  initialises  $\mathcal{F}_{\text{graph}}^{\mathcal{G}}$  to receive its (open) neighbourhood  $\mathcal{N}(u)$ .
  1. **Setup correlated random walks locally.**  
Sample  $\pi_{u,1}, \dots, \pi_{u,\ell-2} \stackrel{\$}{\leftarrow} \mathfrak{S}_B$ , independent uniformly random permutations on  $[\mathcal{N}(u)]$ .
  2. **Prepare executions of  $\Pi_{\text{dir-path}}$ .**

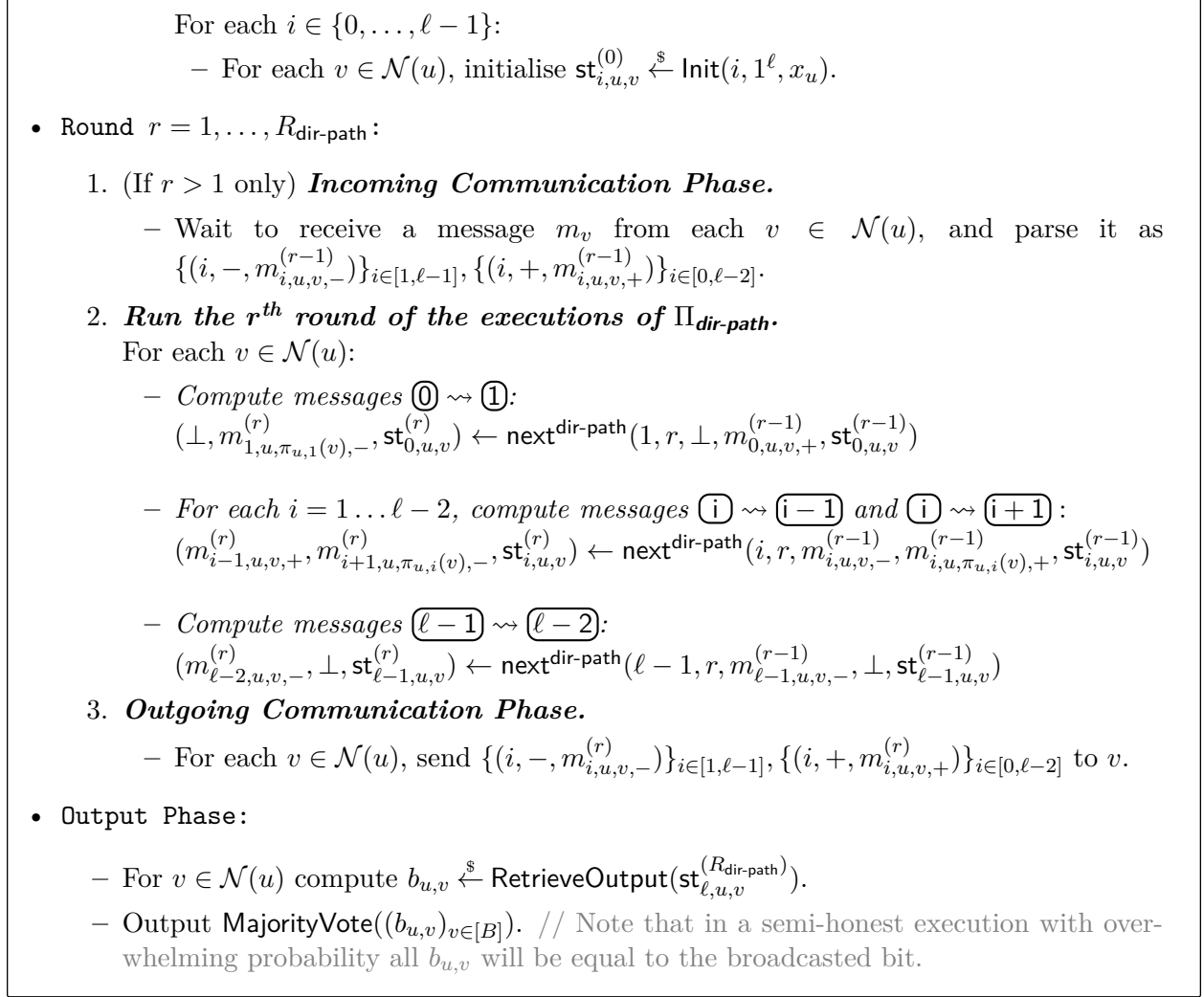


Figure 13: Topology-Hiding OR protocol  $\Pi_{\text{all-graphs}}$  on the class of all connected graphs with at most  $B$  nodes.

**Theorem 6.1** (Topology-hiding OR on all graphs). *Let  $\kappa \in \mathbb{N}$  the statistical security parameter. Let  $B$  be an upper bound on the number of parties, and let  $\ell := 2^{\lceil \log(8\kappa \cdot B^3) \rceil}$ . If  $\Pi_{\text{dir-path}} = (\text{Init}, \text{next}^{\text{dir-path}}, \text{RetrieveOutput})$  is an  $R_{\text{dir-path}}$ -round locally simulatable protocol for securely computing  $(x_0, \dots, x_{\ell-1}) \mapsto \bigvee_{i=0}^{\ell-1} x_i$  on the directed path  $\textcircled{0} \rightarrow \textcircled{1} \dots \rightarrow \textcircled{\ell-1}$  of length  $\ell$  with security against  $\ell - 1$  corruptions, then  $\Pi_{\text{all-graphs}}$  (Fig. 13) securely realises  $\mathcal{F}_{\text{OR}}$  (Fig. 12) in a topology-hiding manner against a static semi-honest adversary corruption up to all but one party.*

*Proof.* We first show that (except with probability negligible in  $\kappa$ ) the parties indeed obtain the correct broadcast bit. Next, we show present a simulator to show that our protocol is indeed topology hiding.

- *Correctness:* At a high level, correctness of  $\Pi_{\text{dir-paths}}$  follows from correctness of  $\Pi_{\text{dir-paths}}$  and the fact that having the parties sample random permutations on their neighbourhoods

(simulated to completion) defines a set of non-interfering random walks which each cover the graph individually with all but negligible probability [ALM20, Lemma 3.14].

The full argument requires some care, since all the executions of  $\Pi_{\text{dir-paths}}$  are run in parallel. Basically, all we have to show is that the messages are correctly routed by the parties, even though they only have their local views of the walks.

In order to be convinced of this, consider the following interpretation of the instructions. If a party  $u$  sends at round  $r$  a message of the form  $(i - 1, +, m)$  to one of its neighbours  $v$ , it should be taken to mean “ $m$  is the round- $r$  message from  $u$  to  $v$  in the unique execution of  $\Pi_{\text{dir-path}}$  along the path where  $u$  is in position  $\overline{i}$ ,  $v$  is in position  $\overline{i-1}$  and  $\pi_{u,i}(v)$  is in position  $\overline{i+1}$ ”. Similarly, if  $u$  sends at round  $r$  a message of the form  $(i + 1, -, m)$  to  $\pi_{u,i}(v)$ , it should be taken to mean “ $m$  is the round- $r$  message from  $u$  to  $\pi_{u,i}(v)$  in the same unique execution. With these indications, correctness follows from a careful inspection.

- *Security:* We prove the theorem in a series of hybrids. We emphasise that this high-level overview does not hide any technical difficulties, and as such this sequence of hybrids shows how the notion of local simulation allows for simple and modular security proofs in the context of topology-hiding computation.

Before starting with the simulation, observe the following:

- Let  $G = (V, E)$  be the actual underlying graph. Then, the number of paths that are generated is equal to  $L := 2 \cdot |E|$ , since edge node  $u$  starts one path for each  $v \in \mathcal{N}(u)$ .
- Except with probability negligible in  $\kappa$ , each path covers the whole graph, we can thus assume that there is at least one honest party present in each path.

The intuition behind our proof is quite simple: If the adversary corrupts a number of subsequent parties on one path, then there is nothing to hide from the adversary regarding the topology of this path. On the other hand, if there are at least one honest parties on a path between two corrupted parties, then we can use the local simulatability of the connected components left and right of the corrupted parties to show that an adversary cannot distinguish whether the two parties are, say, party  $i - 1$  and  $i + 1$  respective to the same path or respective to two different paths<sup>11</sup>.

Initially, we assume that the simulator has the information to perfectly generate the real view of the parties. We will show in a series of hybrids how to obtain a simulator that generates a view that is indistinguishable from the initial view, only given the inputs and outputs of the corrupted parties.

Considered the paths to be labeled by 1 to  $L^2$  (e.g., ordered by the identity of the first two nodes of the path). Further, let  $I \subset V$  be the parties corrupted by the adversaries. For  $j = 1, \dots, L^2$ , let  $(x_{0,j}, \dots, x_{\ell-1,j})$  be the inputs to  $\Pi_{\text{dir-path}}$  in the  $j$ -th execution (as induced by the permutations  $\pi_{u,1}, \dots, \pi_{u,\ell-2}$  for  $u \in V$ ). Further, let  $I_j \subset \{0, \dots, \ell - 1\}$  denote the parties in the  $j$ -th execution that correspond to nodes  $I \in V$ . (Recall that  $|I_j| < \ell$  for all  $j$ , except with probability negligible in  $\kappa$ .)

---

<sup>11</sup>The fact that *one* honest party between two corrupted parties is sufficient to hide whether the two corrupted parties execute the protocol respective to the same path or two different paths is crucial for the proof carrying over to  $\text{KT}_0$ .

- Hybrid 1:  $\mathcal{S}$  knows the entire graph and every party's input, and simulates the real world exactly, by sampling permutations  $\pi_{u,1}, \dots, \pi_{u,\ell-2}$  for each  $u \in V$  and outputs

$$\left( \prod_{\text{all-graphs}} (\{x_i\}_{i \in V}), \{\pi_{u,i}\}_{u \in I, i \in [\ell-2]}, \bigcup_{j=1}^{L^2} \left\{ \text{View}_i^{\text{dir-path}}(x_{0,j}, \dots, x_{\ell-1,j}; r_{0,j}, \dots, r_{\ell-1,j}) \right\}_{i \in I_j} \right).$$

(Note that since we assume to be in the semi-honest setting and each party chooses fresh randomness for each execution, this indeed perfectly simulates the real view.)

- Hybrid  $\iota$ , for  $\iota = 0 \dots L^2$ : Let  $b_{\text{OR}}$  the output of the functionality  $\mathcal{F}_{\text{OR}}$ . Let  $I_{j,1}, \dots, I_{j,\mu_j}$  be a partitioning of  $I_j$  into pairwise disconnected components such that each  $I_{j,k}$  is connected (i.e., a partitioning into maximally connected components). Further, for each  $k \in [\mu_j]$  let  $x_{0,j}^{(k)}, \dots, x_{\ell-1,j}^{(k)}$  be chosen such that:

- $x_{0,j}^{(k)} = x_{0,j}$  for all  $j \in I_{j,k}$
- $x_{0,j}^{(k)}$  is chosen independently from  $x_{0,j}$  for all  $j \notin I_{j,k}$ , conditioned on  $\bigvee_{i=0}^{\ell-1} x_{i,j}^{(k)} = b_{\text{OR}}$ .

We define the  $\iota$ -th hybrid as follows:

$$\left( \prod_{\text{all-graphs}} (\{x_i\}_{i \in V}), \{\pi_{u,i}\}_{u \in I, i \in [\ell-2]}, \bigcup_{j=1}^{\iota} \bigcup_{k=1}^{\mu_k} \left\{ \text{View}_i^{\text{dir-path}}(x_{0,j}^{(k)}, \dots, x_{\ell-1,j}^{(k)}; r_{0,j}^{(k)}, \dots, r_{\ell-1,j}^{(k)}) \right\}_{i \in I_{j,k}}, \bigcup_{j=\iota+1}^{L^2} \left\{ \text{View}_i^{\text{dir-path}}(x_{0,j}, \dots, x_{\ell-1,j}; r_{0,j}, \dots, r_{\ell-1,j}) \right\}_{i \in I_j} \right).$$

The indistinguishability from Hybrid  $\iota$  to  $\iota + 1$  follows directly by Lemma Lemma 4.3 (since all executions use independent randomness, and the lemma holds for arbitrary inputs).

- Hybrid  $L^2$ : Note that in hybrid  $L^2$  the view of the adversary is simulated as

$$\left( \prod_{\text{all-graphs}} (\{x_i\}_{i \in V}), \{\pi_{u,i}\}_{u \in I, i \in [\ell-2]}, \bigcup_{j=1}^{L^2} \bigcup_{k=1}^{\mu_k} \left\{ \text{View}_i^{\text{dir-path}}(x_{0,j}^{(k)}, \dots, x_{\ell-1,j}^{(k)}; r_{0,j}^{(k)}, \dots, r_{\ell-1,j}^{(k)}) \right\}_{i \in I_{j,k}} \right),$$

for which the following holds:

- The distribution is *independent* of the inputs of the honest parties, since the honest parties  $x_{0,j}$  are replaced by simulated inputs  $x_{0,j}^{(k)}$  in the  $j$ -th hybrid (which only requires the knowledge of the input of the corrupted parties together with the output  $b_{\text{OR}}$ ).
- The distribution is *independent* of the graph topology outside the view of the adversary, since the view is generated for each tuple  $(j, k)$  with fresh randomness. Further, since  $I_{j,k}$  is by construction connected, the adversary *knows* the topology withing  $I_{j,k}$ .

By the above considerations, given a semi-honest adversary  $\mathcal{A}$ , there exists a simulator  $\mathcal{S}$  that generates the view of Hybrid  $L^2$  (which is indistinguishable to the real distribution) solely given access to  $\mathcal{F}_{\text{graph-info}}(\mathcal{F}_{\text{OT}})$ . This concludes the proof. □

As an immediate corollary of the proof of Theorem 6.1, we obtain a black-box compiler for locally simulatable protocol for  $\mathcal{F}_{\text{OR}}$  from directed paths to any topology. This is simply due to the observation that the simulator described above is local. Note though that for the task of obtaining locally-simulatable OT, one can replace the correlated random walks by a fixed covering walk<sup>12</sup>, since for that purpose the topology does not need to be hidden.

**Corollary 6.2** (Locally simulatable OR on any graph). *Let  $G$  be a graph. Assuming the existence of a secure 2-party computation protocol with constant rounds and constant overhead, there exists a locally simulatable protocol for securely computing the  $\mathcal{F}_{\text{OR}}$  functionality in the presence of a semi-honest adversary corrupting any number of parties.*

Going from THB to general THC can be achieved via standard techniques, which we briefly recall in the following. On a high level, given topology hiding broadcast the parties can first decide on an enumeration  $1, \dots, |V|$  of the parties (this can be achieved, e.g., by each party broadcasting a string in a sufficiently large interval and sorting the parties based on the lexicographic order of the strings). Given this enumeration, the parties can set up point to point channels using any key exchange protocol (which, in particular, is implied by oblivious transfer). Finally, given these topology-hiding point-to-point channels, the parties can execute any MPC protocol to achieve general topology-hiding secure computation. We therefore obtain the following corollary.

**Corollary 6.3** (THC on all graphs). *Assuming the existence of a secure 2-party computation protocol with constant rounds and constant overhead, there exists a protocol for securely computing any efficiently computable functionality against a semi-honest adversary corrupting all-but-one parties, where only the total number of parties in the graph is leaked (assuming a known a priori bound on the number of parties).*

**Acknowledgments.** We thank Elette Boyle, Ran Cohen, and Tal Moran for helpful discussions. L. Kohl is funded by NWO Gravitation project QSC. Pierre Meyer was supported by ERC Project HSS (852952).

## Bibliography

- [AGPV88] Baruch Awerbuch, Oded Goldreich, David Peleg, and Ronen Vainish. A tradeoff between information and communication in broadcast protocols. In John H. Reif, editor, *VLSI Algorithms and Architectures*, pages 369–379, New York, NY, 1988. Springer New York.
- [ALM17] Adi Akavia, Rio LaVigne, and Tal Moran. Topology-hiding computation on all graphs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology –*

---

<sup>12</sup>A *walk* in a graph is an alternating sequence of adjacent vertices and edges; both vertices and edges may be repeated. A *covering walk* contains each vertex at least once.

- CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 447–467, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.
- [ALM20] Adi Akavia, Rio LaVigne, and Tal Moran. Topology-hiding computation on all graphs. *Journal of Cryptology*, 33(1):176–227, January 2020.
- [AM17] Adi Akavia and Tal Moran. Topology-hiding computation beyond logarithmic diameter. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 609–637, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.
- [BBC<sup>+</sup>19] Marshall Ball, Elette Boyle, Ran Cohen, Tal Malkin, and Tal Moran. Is information-theoretic topology-hiding computation possible? In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 502–530, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany.
- [BBC<sup>+</sup>20] Marshall Ball, Elette Boyle, Ran Cohen, Lisa Kohl, Tal Malkin, Pierre Meyer, and Tal Moran. Topology-hiding communication from minimal assumptions. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 473–501, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany.
- [BBMM18] Marshall Ball, Elette Boyle, Tal Malkin, and Tal Moran. Exploring the boundaries of topology-hiding computation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 294–325, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.
- [BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, IL, USA, May 2–4, 1988. ACM Press.
- [BM90] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO’89*, volume 435 of *Lecture Notes in Computer Science*, pages 547–557, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.
- [Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.
- [CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 11–19, Chicago, IL, USA, May 2–4, 1988. ACM Press.
- [DDN14] Bernardo David, Rafael Dowsley, and Anderson C. A. Nascimento. Universally composable oblivious transfer based on a variant of LPN. In Dimitris Gritzalis, Aggelos



- Kiayias, and Ioannis G. Askoxylakis, editors, *CANS 14: 13th International Conference on Cryptology and Network Security*, volume 8813 of *Lecture Notes in Computer Science*, pages 143–158, Heraklion, Crete, Greece, October 22–24, 2014. Springer, Heidelberg, Germany.
- [DGH<sup>+</sup>20] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Daniel Masny, and Daniel Wichs. Two-round oblivious transfer from CDH or LPN. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 768–797, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.
- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.
- [Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. *Cryptology ePrint Archive*, Report 2000/063, 2000. <https://eprint.iacr.org/2000/063>.
- [Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.
- [HMTZ16] Martin Hirt, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Network-hiding communication and applications to multi-party protocols. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 335–365, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.
- [IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 433–442, Victoria, BC, Canada, May 17–20, 2008. ACM Press.
- [Li22] Shuaishuai Li. Towards practical topology-hiding computation. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part I*, volume 13791 of *Lecture Notes in Computer Science*, pages 588–617, Taipei, Taiwan, December 5–9, 2022. Springer, Heidelberg, Germany.
- [LZM<sup>+</sup>18] Rio LaVigne, Chen-Da Liu Zhang, Ueli Maurer, Tal Moran, Marta Mularczyk, and Daniel Tschudi. Topology-hiding computation beyond semi-honest adversaries. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part II*, volume 11240 of *Lecture Notes in Computer Science*, pages 3–35, Panaji, India, November 11–14, 2018. Springer, Heidelberg, Germany.
- [MOR15] Tal Moran, Ilan Orlov, and Silas Richelson. Topology-hiding computation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 159–181, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

- [MST03] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On e-biased generators in NC0. In *44th Annual Symposium on Foundations of Computer Science*, pages 136–145, Cambridge, MA, USA, October 11–14, 2003. IEEE Computer Society Press.
- [OW14] Ryan ODonnell and David Witmer. Goldreich’s prg: evidence for near-optimal polynomial stretch. In *2014 IEEE 29th Conference on Computational Complexity (CCC)*, pages 1–12. IEEE, 2014.
- [RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st Annual ACM Symposium on Theory of Computing*, pages 73–85, Seattle, WA, USA, May 15–17, 1989. ACM Press.
- [Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.
- [YZ16] Yu Yu and Jiang Zhang. Cryptography with auxiliary input and trapdoor from constant-noise LPN. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 214–243, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.