Towards Topology-Hiding Computation from Oblivious Transfer

Marshall Ball[1], Alexander Bienstock[1], Lisa Kohl[2], and Pierre Meyer[3](✉)

[1] New York University, New York, USA
marshall.ball@cs.nyu.edu, abienstock@cs.nyu.edu
[2] CWI, Cryptology Group, Amsterdam, The Netherlands
lisa.kohl@cwi.nl
[3] IDC Herzliya, ISRAEL and IRIF, Université Paris Cité, CNRS, Paris, France
pierre.meyer@irif.fr

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 1980 s 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 communication 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).

The full version [BBKM23] is available as entry 2023/849 in the IACR eprint archive.

© International Association for Cryptologic Research 2023
G. Rothblum and H. Wee (Eds.): TCC 2023, LNCS 14369, pp. 349–379, 2023.
https://doi.org/10.1007/978-3-031-48615-9_13

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 network, securely computing a function using 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.[1] 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 communciation 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].

[1] 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.

- **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*,[2] THC is possible for graphs of *constant* degree and *logarithmic* diameter [MOR15, BBMM18].[3]

  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+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.[4] 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:

Theorem 1 (Topology-Hiding Computation on All Graphs,Informal). *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.*

[2] 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.

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

[4] 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+20] showed that OT is necessary for topology-hiding *anonymous broadcast* on even simpler graphs.

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+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" KT_0 [AGPV88]), where parties only know pseudonomyms 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 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+20], or on the computational Diffie-Hellman (CDH) assumption [BM90, DGH+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]).

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+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.

2 Technical Overview

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

Theorem 2 (Topology-Hiding Computation on All Graphs, Informal). *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 ①-②-③-④-⑤-⑥-⑦-⑧ (where parties ②, ③, ⑥, and ⑦ are corrupt), the views of parties {②, ③} and {⑥, ⑦} should be simulated independently. Intuitively, this means the adversary cannot tell if {②, ③} and {⑥, ⑦} are participating in the same protocol or, *e.g.* in two different executions ①-②-③-④-Ⓐ-Ⓑ-Ⓒ-Ⓓ and Ⓔ-Ⓕ-Ⓖ-Ⓗ-⑤-⑥-⑦-⑧. 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 obliviously 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].

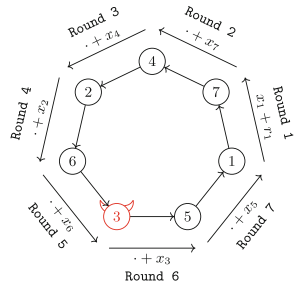
We now expand (still at a high level) on each of these three points, without assuming familiarity with topology-hiding computation.

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).

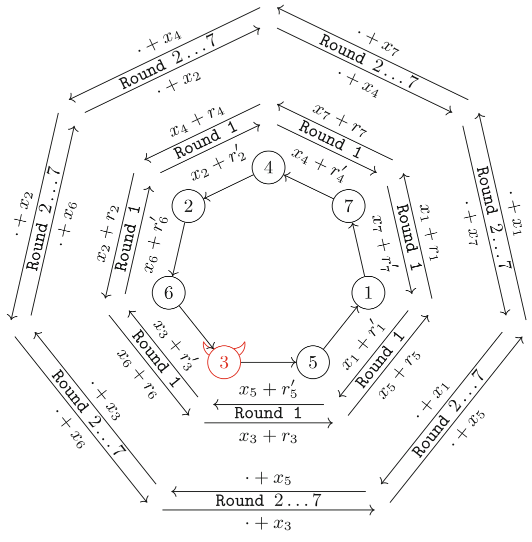
Our starting point is the observation that many topology-hiding protocols can be described informally in a very modular fashion, and yet their formal description (and the corresponding security proof) are inaccessibly monolithic. We start by a gentle introduction to this concept, with a modular presentation 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)[5]. 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 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+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 Fig. 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

[5] In fact, the protocol we describe can be seen as a conceptually simpler alternative to Ball et al.'s [BBC+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_7', then r_7 and r_4', then r_1 and r_2', and finally r_5 and r_6'.

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

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 Fig. 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-neighboring 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[6]. This allows a randomly chosen[7] party to

[6] 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.

[7] The stationary distribution is not uniform, but nevertheless each party has non-negligible probability of being the last party.

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[8].

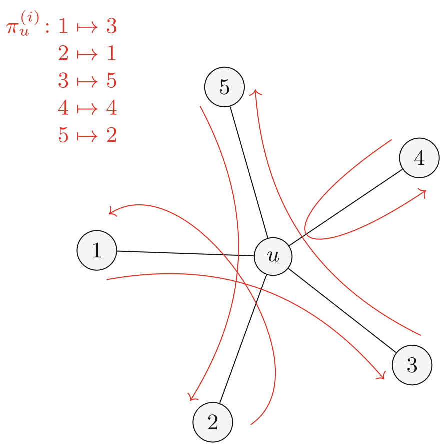
Akavia et al.'s [ALM17, ALM20] elegant solution is to have each party initially send a box to each of their neighbors. In every subsequent round, each party takes all the boxes it just received (one per neighbor), plugs in their input, then shuffles all boxes and sends one to each neighbor. After T rounds, 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+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.

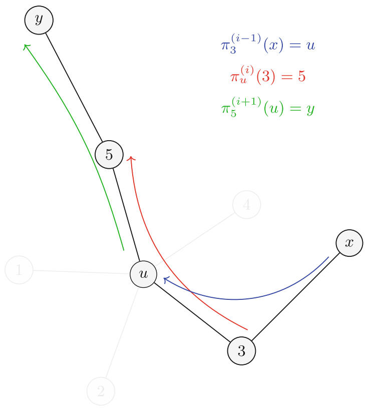
Takeway for Our Protocol: Abstracting out, Akavia et al.'s [ALM17, ALM20] protocol can be seen as first having each party sample T permutations on their neighborhood (as illustrated in Fig. 2, this globally define a mesh of random walks, where each party knows only their position), and then having the 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.

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 able to tell when they are participating in the same execution (i.e. in the same walk) or not. To

[8] 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.



(a) Visual representation of the i^{th} permutation on ⓤ's neighborhood ($\pi_u^{(i)}$), defining ⓤ's neighborhood in five distinct walks.



(b) The family $(\pi_v^{(j)})_{v \in V, j \in [T]}$ defines a family of correlated[a] random walks. Each party knows their position in the walk (*e.g.* party ⓤ knows they are the i^{th} party in the highlighted walk, but they have no way of identifying if this walk is the same as another one in which ⓤ also knows its position).

[a]The walks are *correlated* in the sense that no two walks borrow the same edge at the same time.

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

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 $\mathcal{Z}_1, \mathcal{Z}_2, \ldots$ 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[9]:

[9] However our final protocol will turn out to be both topology-hiding and locally simulatable.

- *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 random string (in a topology-hidding 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^{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.

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 emulates 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 Fig. 3) and then explain the primary ways in which the protocol enables proper topology-hiding emulation and local simulatability. In the full version of this paper [BBKM23], we note some differences between our protocol and the protocol of [MOR15].

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 $\widetilde{\mathsf{P}}_i$. Given each party $\widetilde{\mathsf{P}}_i$'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 $\widetilde{\mathsf{P}}_i$ 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) $\mathsf{2PC}$ computations. At a high-level, the first and second $\ell/2$ real parties will emulate the first and second virtual parties, $\mathsf{P}_{0,0}$ and $\mathsf{P}_{0,1}$, respectively, of a $\mathsf{2PC}$ 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 $\mathsf{2PC}$ computation outputs to both virtual parties $\bigvee_{i=0}^{\ell-1} b_i$. Now, in each round of this $\mathsf{2PC}$, virtual party $\mathsf{P}_{0,0}$ is emulated recursively via another $\mathsf{2PC}$ between virtual parties $\mathsf{P}_{1,0}$ and $\mathsf{P}_{1,1}$, which are in turn emulated by, respectively, the first and second $\ell/4$ real parties recursively (and similarly for virtual party $\mathsf{P}_{0,1}$). Virtual party $\mathsf{P}_{0,0}$ is emulated by virtual parties $\mathsf{P}_{1,0}$ and $\mathsf{P}_{1,1}$ as follows: $\mathsf{P}_{1,0}$ and $\mathsf{P}_{1,1}$ 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 $\mathsf{2PC}$) so that $\mathsf{P}_{0,0}$'s input is emulated by $x_{0,0} = (b_0||b_1||\dots||b_{\ell/2-1})$. Similarly, $\mathsf{P}_{1,0}$ and $\mathsf{P}_{1,1}$ each take as input random strings $\widetilde{r}'_{1,0}$ and $\widetilde{r}'_{1,1}$ and combine them (via the $\mathsf{2PC}$) so that $\mathsf{P}_{0,0}$'s random tape is emulated by $\widetilde{r}_{0,0} = \widetilde{r}'_{1,0} \oplus \widetilde{r}'_{1,1}$. Finally, the $\mathsf{2PC}$ between $\mathsf{P}_{1,0}$ and $\mathsf{P}_{1,1}$ outputs $\mathsf{P}_{0,0}$'s next message in its $\mathsf{2PC}$ with $\mathsf{P}_{0,1}$ to $\mathsf{P}_{1,1}$, who then passes it to the first virtual party $\mathsf{P}_{1,2}$ that participates in the $\mathsf{2PC}$ emulating $\mathsf{P}_{0,1}$. Note that *only* virtual parties $\mathsf{P}_{1,1}$ and $\mathsf{P}_{1,2}$ see and pass to each other the messages for the higher-level $\mathsf{2PC}$; 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 $\mathsf{2PC}$'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) $\mathsf{2PC}$'s. Again, briefly, the $\mathsf{2PC}$'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 $\mathsf{2PC}$ 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.

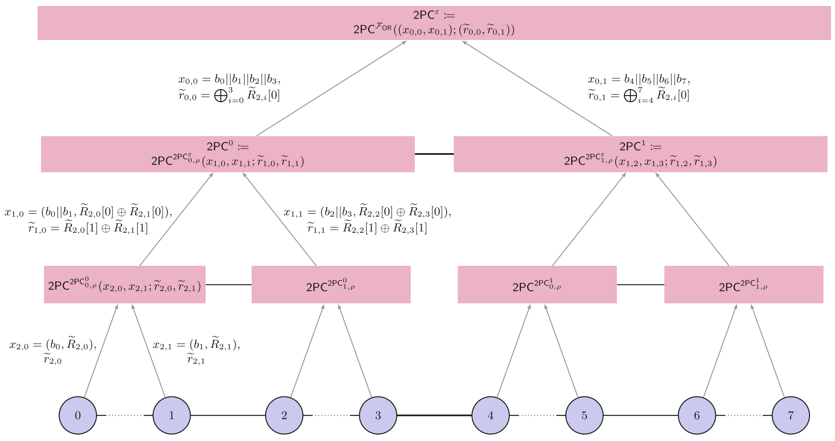

$\mathsf{2PC}^{\varepsilon} :=$
$\mathsf{2PC}^{\mathcal{F}_{\mathsf{OR}}}((x_{0,0}, x_{0,1}); (\widetilde{r}_{0,0}, \widetilde{r}_{0,1}))$
$x_{0,0} = b_0||b_1||b_2||b_3,$
$\widetilde{r}_{0,0} = \bigoplus_{i=0}^{3} \widetilde{R}_{2,i}[0]$
$x_{0,1} = b_4||b_5||b_6||b_7,$
$\widetilde{r}_{0,1} = \bigoplus_{i=4}^{7} \widetilde{R}_{2,i}[0]$
$\mathsf{2PC}^{0} :=$
$\mathsf{2PC}^{\mathsf{2PC}^{\varepsilon}_{0,\rho}}(x_{1,0}, x_{1,1}; \widetilde{r}_{1,0}, \widetilde{r}_{1,1})$
$\mathsf{2PC}^{1} :=$
$\mathsf{2PC}^{\mathsf{2PC}^{\varepsilon}_{1,\rho}}(x_{1,2}, x_{1,3}; \widetilde{r}_{1,2}, \widetilde{r}_{1,3})$
$x_{1,0} = (b_0||b_1, \widetilde{R}_{2,0}[0] \oplus \widetilde{R}_{2,1}[0]),$
$\widetilde{r}_{1,0} = \widetilde{R}_{2,0}[1] \oplus \widetilde{R}_{2,1}[1]$
$x_{1,1} = (b_2||b_3, \widetilde{R}_{2,2}[0] \oplus \widetilde{R}_{2,3}[0]),$
$\widetilde{r}_{1,1} = \widetilde{R}_{2,2}[1] \oplus \widetilde{R}_{2,3}[1]$
$\mathsf{2PC}^{\mathsf{2PC}^{0}_{0,\rho}}(x_{2,0}, x_{2,1}; \widetilde{r}_{2,0}, \widetilde{r}_{2,1})$
$\mathsf{2PC}^{\mathsf{2PC}^{0}_{1,\rho}}$
$\mathsf{2PC}^{\mathsf{2PC}^{1}_{0,\rho}}$
$\mathsf{2PC}^{\mathsf{2PC}^{1}_{1,\rho}}$
$x_{2,0} = (b_0, \widetilde{R}_{2,0}),$
$\widetilde{r}_{2,0}$
$x_{2,1} = (b_1, \widetilde{R}_{2,1}),$
$\widetilde{r}_{2,1}$
0
1
2
3
4
5
6
7

Fig. 3. Depiction of the directed path protocol $\Pi_{\mathsf{dir\text{-}path}}$ for a path of length $\ell = 8$. Each interior node represents a $\mathsf{2PC}$ which gets its inputs and randomness from its two children. This $\mathsf{2PC}$ computes the next message for virtual party P_0 (resp. P_1) in the $\mathsf{2PC}$ 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).

For clarity, we depict an example computation with $\ell = 8$ in Fig. 3. We shall first focus on *real* parties $\widetilde{\mathsf{P}}_0$ and $\widetilde{\mathsf{P}}_1$. 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 $\mathsf{P}_{0,0}$ and (ii) for the emulation of each $\mathsf{2PC}$ in which virtual party $\mathsf{P}_{1,0}$ participates (one for each of the R_{2PC} rounds of the root $\mathsf{2PC}$).

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

The $\mathsf{2PC}$'s which virtual party $\mathsf{P}_{1,0}$ executes with $\mathsf{P}_{1,1}$ correspond to emulations of the next message function of virtual party $\mathsf{P}_{0,0}$ in the highest level $\mathsf{2PC}$, which simply computes the OR of $\mathsf{P}_{0,0}$'s and $\mathsf{P}_{0,1}$'s input bits. $\mathsf{P}_{1,0}$ and $\mathsf{P}_{1,1}$ emulate (via these $\mathsf{2PC}$ executions) $\mathsf{P}_{0,0}$'s input bits as $(b_0||b_1||b_2||b_3)$, and $\mathsf{P}_{0,0}$'s random tape as $\widetilde{r}_{0,0} = \widetilde{r}'_{1,0} \oplus \widetilde{r}'_{1,1}$. Again, recall that, recursively, $\mathsf{P}_{1,0}$ (resp. $\mathsf{P}_{1,1}$)

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

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. 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 infromation 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). In the full version [BBKM23], we 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 Sect. 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.

3 Preliminaries

Notations. For $m < n \in \mathbb{N}$ let $[n] = \{1, \ldots, n\}$ and $[m, n] = \{m, m+1, \ldots, n\}$. In our protocols we sometimes denote by B an upper bound on the number of participating parties. The security parameter is denoted by λ. 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 $i > x, D'[i] = \bot$. Finally, we let $||_{j=i}^{n} x_j = x_i||x_{i+1}|| \ldots ||x_n$

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 $\mathcal{N}_G[v] := \mathcal{N}_G(v) \cup \{v\}$.

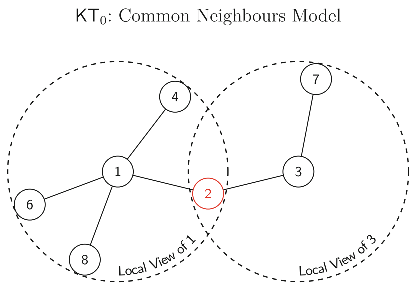
3.1 Topology-Hiding Computation (THC)

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.

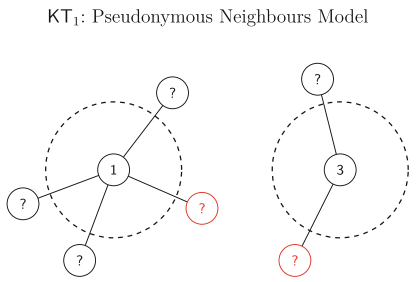
UC Framework. The simulation-based definition is defined in the UC framework of [Can00]. We will consider computationally bounded, static, and semi-honest adversaries and environments.

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 $\mathcal{F}^{\mathcal{G}}_{\mathsf{graph}}$-hybrid model, where $\mathcal{F}^{\mathcal{G}}_{\mathsf{graph}}$ is some functionality that allows parties to communicate with their neighbors in the graph. Many works have used the model of [MOR15], wherein $\mathcal{F}^{\mathcal{G}}_{\mathsf{graph}}$ informs every party P_v of their local neighbourhood by indeed sending $\mathcal{N}_G(v)$ directly to them, and $\mathcal{F}^{\mathcal{G}}_{\mathsf{graph}}$ thereafter facilitates communication from P_v to some other node u, only if u is indeed a neighbor of v. However, [ALM17] instead has $\mathcal{F}^{\mathcal{G}}_{\mathsf{graph}}$ first sample a random injective function $f\colon E \to [n^2]$, labeling each edge with a random (unique) element from $[n^2]$. Next, $\mathcal{F}^{\mathcal{G}}_{\mathsf{graph}}$ informs every party P_v of their local neighbourhood by instead sending them the set of edge labels $L_v := \{f((u, v))\colon (u, v) \in E\}$. $\mathcal{F}^{\mathcal{G}}_{\mathsf{graph}}$ 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 **K**nowledge **T**ill Radius $\boldsymbol{\sigma}$ Model (KT_σ). These two worlds are illustrated in Fig. 4. 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. 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.



(a) In the 'Common Neighbours' model, colluding parties (①,③) know from their local views if they have common neighbours (②).



(b) In the 'Pseudonymous Neighbours' model, colluding parties (①,③) do not know from their local views if they have common neighbours (?).

Fig. 4. Differing views of parties in KT_0 and KT_1.

Simulation-Based THC. Now we are ready to introduce our simulation-baesd topology-hiding computation definition. The real-world protocol is defined in a model where all communication is transmitted via the functionality $\mathcal{F}_{\mathsf{graph}}^{\mathcal{G},\mathsf{KT}_\sigma}$ (described in Fig. 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 KT_σ, 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, $\mathcal{F}_{\mathsf{graph}}^{\mathcal{G},\mathsf{KT}_\sigma}$ receives the network communication graph G from a special graph party $\mathsf{P}_{\mathsf{graph}}$ and makes sure that $G \in \mathcal{G}$. Then, if $\sigma = 0$, it samples a random injective function $f\colon E \to [n^2]$, 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 π with respect to $\mathcal{G}$ as a set of n PPT interactive Turing machines (ITMs) $(\mathsf{P}_1, \ldots, \mathsf{P}_n)$ (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 $\mathcal{F}_{\mathsf{graph}}^{\mathcal{G},\mathsf{KT}_\sigma}$), and produce output.

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

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

Initialization Phase:

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

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

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

Communication Phase:

Input:

– *If* $\sigma = 1$*:* $\mathcal{F}_{\mathsf{graph}}^{\mathcal{G},\mathsf{KT}_1}$ 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, $\mathcal{F}_{\mathsf{graph}}^{\mathcal{G},\mathsf{KT}_0}$ ignores this input.)

– *If* $\sigma = 0$*:* $\mathcal{F}_{\mathsf{graph}}^{\mathcal{G},\mathsf{KT}_0}$ receives from a party P_v a destination/data pair (l, m) where $f(v, w) = l \in L_v$ indicates to $\mathcal{F}_{\mathsf{graph}}^{\mathcal{G},\mathsf{KT}_0}$ the neighbour w, and m is the message P_v wants to send to P_w. (If $v \notin V$ or $\nexists (v, w) \in E : f(v, w) = l$, $\mathcal{F}_{\mathsf{graph}}^{\mathcal{G},\mathsf{KT}_1}$ ignores this input.)

Output:

– *If* $\sigma = 1$*:* $\mathcal{F}_{\mathsf{graph}}^{\mathcal{G},\mathsf{KT}_1}$ gives output (v, m) to P_w indicating that P_v sent the message m to P_w.

– *If* $\sigma = 0$*:* $\mathcal{F}_{\mathsf{graph}}^{\mathcal{G},\mathsf{KT}_0}$ 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.

Fig. 5. The communication graph functionality (unified definition for KT_0 and 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 KT_0 or 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 $\mathcal{W}^{\mathcal{G},\mathsf{KT}_\sigma}_{\mathsf{graph\text{-}info}}(\mathcal{F})$, that runs internally a copy of the functionality $\mathcal{F}$. The wrapper $\mathcal{W}^{\mathcal{G},\mathsf{KT}_\sigma}_{\mathsf{graph\text{-}info}}(\cdot)$ 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 $\mathsf{P}_{\mathsf{graph}}$, 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 P_v responds with its local neighbourhood information (just like $\mathcal{F}^{\mathcal{G},\mathsf{KT}_\sigma}_{\mathsf{graph}}$). All other input messages from a party P_v are forwarded to $\mathcal{F}$ and every message from $\mathcal{F}$ to a party P_v is delivered to its recipient (Fig. 6).

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

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

Initialization Phase:

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

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

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

Communication Phase:

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

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

Fig. 6. The graph-information wrapper functionality (unified definition for KT_0 and 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 1 (Topology-hiding computation). *We say that a protocol π 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 π securely realizes $\mathcal{W}^{\mathcal{G},\mathsf{KT}_0}_{\mathsf{graph\text{-}info}}(\mathcal{F})$ in the $\mathcal{F}^{\mathcal{G},\mathsf{KT}_0}_{\mathsf{graph}}$-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 Fig. 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 $\mathcal{F}_{\mathsf{bc}}(\mathsf{P}_i)$.

Functionality $\mathcal{F}_{\mathsf{bc}}(\mathsf{P}_i)$

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

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

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.

Fig. 7. The broadcast functionality

Definition 2 (t-THB). *Let $\mathcal{G}$ be a family of graphs and let t be an integer. A protocol π is a* t-THB protocol with respect to $\mathcal{G}$ *if $\pi(\mathsf{P}_v)$ securely realizes $\mathcal{F}_{\mathsf{bc}}(\mathsf{P}_v)$ 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 (Stateless Two-Party Computation Syntax). *A R_{2PC}-round* Stateless Two-Party Computation (2PC) protocol $\mathsf{2PC}^{\mathcal{F}}(x_0, x_1; r_0, r_1) :=(\mathsf{2PC}^{\mathcal{F}}_{0,i}, \mathsf{2PC}^{\mathcal{F}}_{1,i})_{i\in[0,R_{2PC}-1]}$ *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* $\mathsf{2PC}^{\mathcal{F}}_{0,i}(x_0, \{m_{1,j}\}_{j<i}; r_0)$ (resp. $\mathsf{2PC}^{\mathcal{F}}_{1,i}(x_1, \{m_{0,j}\}_{j\leq i}; r_1)$) *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* $\mathsf{2PC}^{\mathcal{F}}_{0,R_{\mathsf{2PC}}-1}(x_0, \{m_{1,j}\}_{j<R_{\mathsf{2PC}}-1}; r_0)$ (resp. $\mathsf{2PC}^{\mathcal{F}}_{1,R_{\mathsf{2PC}}-1}(x_1, \{m_{0,j}\}_{j\leq R_{\mathsf{2PC}}-1}; r_1)$) *additionally gives output y_0 (resp.* only *gives output y_1).*

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

We defer the standard real/ideal world security definition of $\mathsf{2PC}$ with respect to a semi-honest adversary to the real version [BBKM23].

Constant-Overhead Constant-Round $\mathsf{2PC}$. For this work, we need to use a $\mathsf{2PC}$ with constant overhead and constant round complexity. More precisely, we require that $\mathsf{2PC}$ satisfies the following properties: First, for any given functionality $\mathcal{F}$ and the corresponding circuit $\mathcal{C}_{\mathcal{F}}$ that computes it, $\mathsf{2PC}$ has computational (and thus also communication) overhead $O(|\mathcal{C}_{\mathcal{F}}|) + \mathrm{poly}(\lambda)$, 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.

3.3 Efficiently Invertible from Local Information Functionalities

In the full version [BBKM23], we define special *efficiently invertible from local information* functionalities for which we can prove local simulatability of our path protocol (the OR functionality being one example).

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 $f\colon \mathcal{X}^0 \times \cdots \times \mathcal{X}^{\ell-1} \rightarrow \mathcal{Y}^0 \times \cdots \times \mathcal{Y}^{\ell-1}$ in Fig. 8.

We define local simulatability relative to a communication network $G = (V, E)$, where $V = \{0, \ldots, \ell-1\}$, 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.

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 $i \in \{0, \dots, \ell - 1\}$).

Fig. 8. Functionality $\mathcal{F}_f$ for computing $f\colon \mathcal{X}^0 \times \dots \times \mathcal{X}^{\ell-1} \to \mathcal{Y}^0 \times \dots \times \mathcal{Y}^{\ell-1}$.

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 Π 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 Π is of the form

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

where $\Pi(x_0, \dots, x_{\ell-1})$ denotes the outputs of parties $P_0, \dots, P_{\ell-1}$ after the execution of Π 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 $\mathsf{Sim} = (\mathsf{Sim}_1, \dots, \mathsf{Sim}_\mu)$ be a tuple of algorithms[10], such that for each $j \in \{1, \dots, \mu\}$ the following holds:

– Sim_j is a PPT algorithm,
– Sim_j obtains an input/ output pair (x_i, y_i) for all $i \in I_j$,
– Sim_j outputs a simulated view of parties $\{P_i\}_{i \in I_j}$.

[10] Note that the distinction into μ different simulators instead of μ copies of the same simulator is solely for the sake of clarity.

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

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

Definition 4 (Local Simulation). *Let Π be a protocol on G. We say that* Π 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 $I \subset \{0,\ldots,\ell-1\}$ with $|I| \le t$ and for every partitioning of I into pairwise disconnected components $I_1,\ldots,I_\mu$, there exists a PPT simulator $\mathit{Sim} = (\mathit{Sim}_1,\ldots,\mathit{Sim}_\mu)$, such that for all $x_0,\ldots,x_{\ell-1} \in \{0,1\}^\star$ it holds*

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

4.1 Locally Simulatable Protocols Are Execution-Oblivious

In this section we first define *execution obliviousness.* Then, in the full version of the paper [BBKM23], 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.

In the following we restrict to protocols implementing deterministic functionalities with perfect correctness, i.e. for which $\Pi(x_0,\ldots,x_{\ell-1})$ 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 5 (Execution obliviousness.). *Let $G = (V,E)$ be a graph with $V = \{0,\ldots,\ell-1\}$ and let Π be an ℓ-party protocol on G. We say Π is* execution oblivious on G tolerating t corruptions, *if for all sets $I \subseteq \{0,\ldots,\ell-1\}$ with $|I| \le t$ and for any partitioning of I into pairwise disconnected components $I_1,\ldots,I_\mu$ the following holds:*

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

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

it holds:

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

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

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

We defer the proof of this lemma to the full version [BBKM23].

5 Locally Simulatable Protocol for Directed Paths

In this section, we formally present the protocol for computing on a directed path some functionality $\mathcal{F}$ that is *efficiently invertible from local information*. An example of such a functionality is $\mathcal{F}_{\mathsf{OR}}$, which we will use to impelement THB in the next section. We refer the reader back to Sect. 2.2 for a detailed overview of the protocol. Due to space limitations, we defer the proof of local simulatability to the full version of this paper [BBKM23].

5.1 The Path Protocol

The directed path protocol $\Pi_{\mathsf{dir\text{-}path}}$ is formally presented in Fig. 9. As described in Sect. 2.2, the protocol works over a directed path $\mathtt{Path}_\ell = \textcircled{0} \rightarrow \textcircled{1} \cdots \rightarrow \boxed{\ell\text{-}1}$ 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 $\widetilde{\mathsf{P}}_j$. The protocol recursively computes the given functionality $\mathcal{F}$. Recall that $\mathcal{F}$ must be *efficiently invertible from local information*, such as $\mathcal{F}_{\mathsf{OR}}$, which on input bits b_j from each party $\widetilde{\mathsf{P}}_j$, outputs $\bigvee_{j=0}^{\ell-1} b_j$ to every party. When computation of $\mathcal{F}_{\mathsf{OR}}$ used in our higher-level THB protocol of the next section, the input of party $\widetilde{\mathsf{P}}_{j^*}$ corresponding to the broadcaster will be $b_{j^*} = b$, the broadcast bit, and for all $j \neq j^*$, the input of party $\widetilde{\mathsf{P}}_j$ will be $b_j = 0$.

$\Pi_{\mathsf{dir\text{-}path}}$ proceeds by recursively emulating a (constant-round, constant-overhead) $\mathsf{2PC}$ that computes $((y_0||\ldots||y_{\ell/2-1}), (y_{\ell/2}||\ldots||y_{\ell-1})) = \mathcal{F}'((x_0||\ldots||x_{\ell/2-1}), (x_{\ell/2}||\ldots||x_{\ell-1})) = \mathcal{F}(x_0, \ldots, x_{\ell-1})$ 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 $\mathsf{P}_{0,0}$ (and similarly for party $\mathsf{P}_{0,1}$) of the highest-level $\mathsf{2PC}$ is recursively emulated by parties $\widetilde{\mathsf{P}}_0, \ldots \widetilde{\mathsf{P}}_{\ell/2-1}$ on the path by first computing each message that $\mathsf{P}_{0,0}$ sends in this $\mathsf{2PC}$ via another lower-level $\mathsf{2PC}$ between virtual parties $\mathsf{P}_{1,0}$ and $\mathsf{P}_{1,1}$. Parties $\mathsf{P}_{1,0}$ and $\mathsf{P}_{1,1}$ combine their inputs and random strings via this $\mathsf{2PC}$ to emulate $\mathsf{P}_{0,0}$'s input and random tape. $\mathsf{P}_{1,1}$ then receives $\mathsf{P}_{0,0}$'s next message and sends it to $\mathsf{P}_{1,2}$ (the first party emulating $\mathsf{P}_{0,1}$). Continuing in the recursion, both $\mathsf{P}_{1,0}$ and $\mathsf{P}_{1,1}$ are then emulated by another $\mathsf{2PC}$ 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 $\Pi_{\mathsf{dir\text{-}path}}$ 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 $\mathsf{2PC}$ 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 $\mathsf{2PC}$ 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 $\mathsf{2PC}$ functionality $\mathcal{F}$ that the two virtual parties are computing. For the original invocation call, the path considered is the whole path $\mathtt{Path}_\ell = \textcircled{0} \rightarrow \textcircled{1} \cdots \rightarrow \boxed{\ell\text{-}1}$, the recursion depth is $d = 0$, message virtual party is $\sigma = \perp$, output flag is $o = 1$, and the $\mathsf{2PC}$ functionality that will be recursively computed is $\mathcal{F}'$; i.e., on input x_0 from $\mathsf{P}_{0,0}$ (recursively $||_{j=0}^{\ell/2-1} x_j$) and x_1 from $\mathsf{P}_{0,1}$ (recursively $||_{j=\ell/2}^{\ell-1} x_j$), output $\mathcal{F}'(x_0, x_1)$ to $\mathsf{P}_{0,0}$ and $\mathsf{P}_{0,1}$.

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

Efficiency. Recall that we assume the round complexity of the $\mathsf{2PC}$ 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 $\Pi_{\mathsf{dir\text{-}path}}$ is $R[\ell] = 2R_{\mathsf{2PC}} \cdot R[\ell/2] + 2 = \Theta\left(\ell \cdot R_{\mathsf{2PC}}^{\log \ell}\right)$, which is $O(\ell^2)$. Furthermore, each real party on the path executes $R_{\mathsf{2PC}}^{\log \ell - 1}$ $\mathsf{2PC}$'s. The overhead of the highest-level $\mathsf{2PC}$ is $c \cdot |\mathcal{C}_{\mathcal{F}'}| + \text{poly}(\lambda)$, the overhead of the $\mathsf{2PC}$'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 $\mathsf{2PC}$'s executed by the real parties is $O(\ell \cdot (|\mathcal{C}_{\mathcal{F}'}| + \text{poly}(\lambda)))$. Therefore, the total overhead of $\Pi_{\mathsf{dir\text{-}path}}$ is $O(R_{\mathsf{2PC}}^{\log \ell - 1} \cdot \ell \cdot (|\mathcal{C}_{\mathcal{F}'}| + \text{poly}(\lambda))) = O(\ell^2 \cdot (|\mathcal{C}_{\mathcal{F}'}| + \text{poly}(\lambda)))$.

Protocol $\Pi_{\textsf{dir-path}}$

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

The Protocol:

- `Initialisation`: Each party $\widetilde{\mathsf{P}}_j$ receives $(j, x_j, \widetilde{R}_j, M_j, \widetilde{\mathsf{P}}_{j-1}, \widetilde{\mathsf{P}}_{j+1})$ as input, either from the original invocation, or from a recursive call. Each party $\widetilde{\mathsf{P}}_j$ samples $\widetilde{r}_j \xleftarrow{\$} \{0,1\}^\lambda$ and sets $\widetilde{R}_j[d] \leftarrow \widetilde{r}_j$.
- Base Case ($\ell = 2$): Parties $\widetilde{\mathsf{P}}_0$ and $\widetilde{\mathsf{P}}_1$ directly compute the R_{2PC} round protocol $\mathsf{2PC}^{\mathcal{F}}((x_0, \widetilde{R}_0[:\ d-2], \widetilde{R}_0[d-1], M_0), (x_1, \widetilde{R}_1[:\ d-2], \widetilde{R}_1[d-1], M_1); \widetilde{r}_0, \widetilde{r}_1)$. If $\sigma \neq \perp$, then $\widetilde{\mathsf{P}}_\sigma$ 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 $\rho = 0, \ldots, R_{\mathsf{2PC}} - 2$:
  - `For` $k \in [0,1]$`, rounds` $(2\rho+k)\cdot(R[\frac{\ell}{2}]+1)$ `to` $(2\rho+k+1)\cdot(R[\frac{\ell}{2}]+1)-1$`:`

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

    1. Parties $\widetilde{\mathsf{P}}_{k\cdot\ell/2}, \ldots, \widetilde{\mathsf{P}}_{k\cdot\ell/2+\ell/2-1}$ recursively call $\Pi_{\textsf{dir-path}}$ with:
       * *Parameters:* $\boxed{\mathsf{k}\cdot\ell/2} \to \boxed{\mathsf{k}\cdot\ell/2+1} \cdots \to \boxed{\mathsf{k}\cdot\ell/2+\ell/2\text{-}1}$ of length $\ell/2 = 2^{l-1}$, protocol $\mathsf{2PC}$, recursion depth $d+1$, message virtual party $\sigma' = 1-k$, output flag $o = 0$, 2PC functionality $\mathcal{F}_{\mathsf{k}} :=$
         - $\mathsf{P}_{k,0}$'s Input: $x_{k,0}, \widetilde{R}_{k,0}, \widetilde{r}_{k,0}, M_{k,0}$

$$\Big(\text{recursively: } ||_{j=k\cdot\ell/2}^{k\cdot\ell/2+\ell/4-1} x_j, \left\{\bigoplus_{j=k\cdot\ell/2}^{k\cdot\ell/2+\ell/4-1} \widetilde{R}_j[d']\right\}_{d'\in[0,d-1]}, \bigoplus_{j=k\cdot\ell/2}^{k\cdot\ell/2+\ell/4-1} \widetilde{r}_j, M'_{k\cdot\ell/2};$$
$$\text{where for } k=1, M'_{k\cdot\ell/2} = \{m_{0,q}\}_{q<\rho}, \text{ i.e., messages sent so far by } \mathsf{P}_0;$$
$$\text{for } k=0, M'_{k\cdot\ell/2} = M_{k\cdot\ell/2}\Big)$$

         - $\mathsf{P}_{k,1}$'s Input: $b_{k,1}, \widetilde{R}_{k,1}, \widetilde{r}_{k,1}, M_{k,1}$

$$\Big(\text{recursively: } ||_{j=k\cdot\ell/2+\ell/4}^{k\cdot\ell/2+\ell/2-1} x_j, \left\{\bigoplus_{j=k\cdot\ell/2+\ell/4}^{k\cdot\ell/2+\ell/2-1} \widetilde{R}_j[d']\right\}_{d'\in[0,d-1]}, \bigoplus_{j=k\cdot\ell/2+\ell/4}^{k\cdot\ell/2+\ell/2-1} \widetilde{r}_j, M'_{k\cdot\ell/2+\ell/2-1};$$
$$\text{where for } k=0, M'_{k\cdot\ell/2+\ell/2-1} = \{m_{1,q}\}_{q<\rho}, \text{ i.e., messages sent so far by } \mathsf{P}_1;$$
$$\text{for } k=1, M'_{k\cdot\ell/2+\ell/2-1} = M_{k\cdot\ell/2+\ell/2-1}\Big)$$

Fig. 9. Protocol $\Pi_{\textsf{dir-path}}$ which on input x_j from each party $\widetilde{\mathsf{P}}_j$ on a directed path, computes $\mathcal{F}(x_0, \ldots, x_{\ell-1}) = (y_0, \ldots, y_{\ell-1})$ and outputs to party $\widetilde{\mathsf{P}}_j$ their output y_j. Note: each party knows their position on the path.

· $\mathsf{P}_{k,k}$'s output: $\perp$
· $\mathsf{P}_{k,1-k}$'s output: $\mathsf{2PC}^{\mathcal{F}}_{k,\rho}((x_{k,0}||x_{k,1},\{\widetilde{R}_{k,0}[d'] \oplus \widetilde{R}_{k,1}[d']\}_{d'\in[0,d-2]},$
$\widetilde{R}_{k,0}[d-1] \oplus \widetilde{R}_{k,1}[d-1], M_{k,0}), M_{k,1}; \widetilde{r}_{k,0} \oplus \widetilde{r}_{k,1})$

$*$ *Inputs:* $\widetilde{\mathsf{P}}_j$ holds $(j \mod \ell/2, b_j, \widetilde{R}_j, M'_j, \widetilde{\mathsf{P}}_{j-1}, \widetilde{\mathsf{P}}_{j+1})$, where for $\widetilde{\mathsf{P}}_{\ell/2-1+k}$, M'_j is as above, otherwise, $M'_j = M_j$.

2. $\widetilde{\mathsf{P}}_{\ell/2-1+k}$ waits to receive $m_{k,\rho}$ as output of the recursive call to $\Pi_{\mathsf{dir\text{-}path}}$.

Send virtual Party P_k ***'s message*** $m_{k,\rho}$ ***to virtual Party*** P_{1-k}***.***

1. $\widetilde{\mathsf{P}}_{\ell/2-1+k}$ sends $m_{k,\rho}$ to $\widetilde{\mathsf{P}}_{\ell/2-k}$.

– For $\rho = R_{\mathsf{2PC}} - 1$ (the last 2PC round):
- `Rounds` $(2R_{\mathsf{2PC}}-2)\cdot(R[\frac{\ell}{2}]+1)$ `to` $(2R_{\mathsf{2PC}}-2)\cdot(R[\frac{\ell}{2}]+1)+R[\frac{\ell}{2}]-1$:
  Compute virtual Party P_0 ***'s last 2PC message*** $m_{0,R_{\mathsf{2PC}}-1}$ ***and output*** y_0***.***
  1. As above, except if $\sigma = 0$, then instead of using message virtual party $\sigma' = 1$ for the recursive call to $\Pi_{\mathsf{dir-path}}$, we use $\sigma' = \sigma$. Furthermore, if $o = 1$, then instead of using output flag $o' = 0$, we use $o' = 1$. Then, if $\sigma = 0$, $\widetilde{\mathsf{P}}_0$ waits to receive the output y_0 from the recursive call to $\Pi_{\mathsf{dir-path}}$ then outputs it to its invoker themself, and sets $m_{0,R_{\mathsf{2PC}}-1} = \perp$. Otherwise, $\widetilde{\mathsf{P}}_{\ell/2-1}$ waits to receive $m_{0,R_{\mathsf{2PC}}-1}$ from the recursive call to $\Pi_{\mathsf{dir-path}}$.
- `Round` $(2R_{\mathsf{2PC}}-1)\cdot(R[\frac{\ell}{2}]+1)-1$:
  Send virtual Party P_0 ***'s last 2PC message*** $m_{0,R_{\mathsf{2PC}}-1}$ ***to Party*** P_1.
  1. As above.
- `Rounds` $(2R_{\mathsf{2PC}}-1)\cdot(R[\frac{\ell}{2}]+1)$ `to` $(2R_{\mathsf{2PC}}-1)\cdot(R[\frac{\ell}{2}]+1)+R[\frac{\ell}{2}]-1$:
  Compute virtual Party P_1 ***'s 2PC output.***
  1. As above, except if $\sigma = 0$, then return $\perp$. Otherwise, if $\sigma = 1$ or $\sigma = \perp$, then instead of using message virtual party $\sigma' = 0$ for the recursive call to $\Pi_{\mathsf{dir-path}}$, we use $\sigma' = \sigma$. Furthermore, if $o = 1$, then instead of using output flag $o' = 0$, we use $o' = 1$. Then, $\widetilde{\mathsf{P}}_{\ell-1}$ waits to receive the output y_1 of the recursive call to $\Pi_{\mathsf{dir-path}}$ (if any) and outputs it to its invoker themself.

Fig. 9. (*continued*)

6 Extension to All Graphs

We refer to the full version of this paper [BBKM23] for how to use our protocol with local simulatability on paths to achieve topology-hiding computation on all graphs, building on the technqiues of [ALM17]. We state the relevant Theorem and Corollaries below.

Theorem 3 (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 $\ell := 2^{\lfloor \log(8\kappa \cdot B^3) \rfloor}$. If $\Pi_{\mathsf{dir\text{-}path}} = (\mathsf{Init}, \mathsf{next}^{\mathsf{dir\text{-}path}}, \mathsf{RetrieveOutput})$ is an $R_{\mathsf{dir\text{-}path}}$-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 $\textcircled{0} \rightarrow \textcircled{2} \cdots \rightarrow \boxed{\ell - 1}$ of length ℓ with security against $\ell - 1$ corruptions, then there exists a protocol that securely realises $\mathcal{F}_{\mathsf{OR}}$ in a topology-hiding manner against a static semi-honest adversary corrupting up to all but one party.*

As an immediate corollary of the proof of Theorem 3 (in the full version [BBKM23]) we obtain a black-box compiler for locally simulatable protocol for $\mathcal{F}_{\mathsf{OR}}$ 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[11], since for that purpose the topology does not need to be hidden.

Corollary 1 (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 $\mathcal{F}_{\mathsf{OR}}$ 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 2 (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*

[11] 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.

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 apriori bound on the number of parties).

Acknowledgments. We thank Elette Boyle, Ran Cohen, and Tal Moran for helpful discussions. Marshall Ball is supported in part by the Simons Foundation. Lisa Kohl is funded by NWO Gravitation project QSC. Pierre Meyer was supported by ERC Project HSS (852952) and by AFOSR Award FA9550-21-1-0046. We thank the anonymous reviewers of TCC for helpful feedback regarding the presentation of our results.

References

[AGPV88] Awerbuch, B., Goldreich, O., Peleg, D., Vainish, R.: A tradeoff between information and communication in broadcast protocols. In: Reif, J.H. (ed.) AWOC 1988. LNCS, vol. 319, pp. 369–379. Springer, New York (1988). https://doi.org/10.1007/BFb0040404

[ALM17] Akavia, A., LaVigne, R., Moran, T.: Topology-hiding computation on all graphs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 447–467. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_15

[ALM20] Akavia, A., LaVigne, R., Moran, T.: Topology-hiding computation on all graphs. J. Cryptol. **33**(1), 176–227 (2020)

[AM17] Akavia, A., Moran, T.: Topology-hiding computation beyond logarithmic diameter. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10212, pp. 609–637. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56617-7_21

[BBC+19] Ball, M., Boyle, E., Cohen, R., Malkin, T., Moran, T.: Is information-theoretic topology-hiding computation possible? In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11891, pp. 502–530. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_20

[BBC+20] Ball, M., et al.: Topology-hiding communication from minimal assumptions. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12551, pp. 473–501. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64378-2_17

[BBKM23] Ball, M., Bienstock, A., Kohl, L., Meyer, P.: Towards topology-hiding computation from oblivious transfer. Cryptology ePrint Archive, Paper 2023/849 (2023). https://eprint.iacr.org/2023/849

[BBMM18] Ball, M., Boyle, E., Malkin, T., Moran, T.: Exploring the boundaries of topology-hiding computation. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 294–325. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_10

[BGW88] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th Annual ACM Symposium on Theory of Computing, pp. 1–10, Chicago, IL, USA, ACM Press, 2–4 May 1988

[BM90] Bellare, M., Micali, S.: Non-interactive oblivious transfer and applications. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 547–557. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_48

[Can00] Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000)

[CCD88] Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th Annual ACM Symposium on Theory of Computing, pp. 11–19, Chicago, IL, USA, ACM Press, 2–4 May 1988

[DDN14] David, B., Dowsley, R., Nascimento, A.C.A.: Universally composable oblivious transfer based on a variant of LPN. In: Gritzalis, D., Kiayias, A., Askoxylakis, I. (eds.) CANS 2014. LNCS, vol. 8813, pp. 143–158. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12280-9_10

[DGH+20] Döttling, N., Garg, S., Hajiabadi, M., Masny, D., Wichs, D.: Two-round oblivious transfer from CDH or LPN. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 768–797. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_26

[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.), 19th Annual ACM Symposium on Theory of Computing, pp. 218–229, New York City, NY, USA, ACM Press, 25–27 May 1987

[Gol00] Goldreich, O.: Candidate one-way functions based on expander graphs. Cryptology ePrint Archive, Report 2000/063 (2000). https://eprint.iacr.org/2000/063

[HMTZ16] Hirt, M., Maurer, U., Tschudi, D., Zikas, V.: Network-hiding communication and applications to multi-party protocols. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 335–365. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_12

[IKOS08] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with constant computational overhead. In: Richard, E.L., Cynthia, D. (eds.), 40th Annual ACM Symposium on Theory of Computing, pp. 433–442, Victoria, BC, Canada, ACM Press, 17–20 May 2008

[Li22] Li, S.: Towards practical topology-hiding computation. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology - ASIACRYPT 2022. ASIACRYPT 2022. LNCS, vol. 13791, pp. 588–617. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22963-3_20

[LZM+18] LaVigne, R., Liu-Zhang, C.-D., Maurer, U., Moran, T., Mularczyk, M., Tschudi, D.: Topology-hiding computation beyond semi-honest adversaries. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018. LNCS, vol. 11240, pp. 3–35. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03810-6_1

[MOR15] Moran, T., Orlov, I., Richelson, S.: Topology-hiding computation. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9014, pp. 159–181. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46494-6_8

[MST03] Mossel, E., Shpilka, A., Trevisan, L.: On e-biased generators in NC0. In 44th Annual Symposium on Foundations of Computer Science, pp. 136–145, Cambridge, MA, USA, IEEE, Computer Society Press, 11–14 October 2003

[OW14] ODonnell, R., Witmer, D.: Goldreich's prg: evidence for near-optimal polynomial stretch. In: 2014 IEEE 29th Conference on Computational Complexity (CCC), pp. 1–12. IEEE (2014)

[RB89] Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: 21st Annual ACM Symposium on Theory of Computing, pp. 73–85, Seattle, WA, USA, ACM Press, 15–17 May 1989

[Yao82] Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, pp. 160–164, Chicago, Illinois, IEEE Computer Society Press, 3–5 November 1982

[YZ16] Yu, Yu., Zhang, J.: Cryptography with auxiliary input and trapdoor from constant-noise LPN. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 214–243. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_9