Topology-Hiding Computation Beyond Semi-Honest Adversaries

Rio LaVigne[1⋆], Chen-Da Liu-Zhang[2], Ueli Maurer[2], Tal Moran[3⋆⋆], Marta Mularczyk[2⋆⋆⋆], and Daniel Tschudi[4†]

[1] rio@mit.edu, MIT
[2] {lichen,maurer,mumarta}@inf.ethz.ch, ETH Zurich
[3] talm@idc.ac.il, IDC Herzliya
[4] tschudi@cs.au.dk, Aarhus University

Abstract. Topology-hiding communication protocols allow a set of parties, connected by an incomplete network with unknown communication graph, where each party only knows its neighbors, to construct a complete communication network such that the network topology remains hidden even from a powerful adversary who can corrupt parties. This communication network can then be used to perform arbitrary tasks, for example secure multi-party computation, in a topology-hiding manner.

Previously proposed protocols could only tolerate passive corruption. This paper proposes protocols that can also tolerate fail-corruption (i.e., the adversary can crash any party at any point in time) and so-called semi-malicious corruption (i.e., the adversary can control a corrupted party's randomness), without leaking more than an arbitrarily small fraction of a bit of information about the topology. A small-leakage protocol was recently proposed by Ball et al. [Eurocrypt'18], but only under the unrealistic set-up assumption that each party has a trusted hardware module containing secret correlated pre-set keys, and with the further two restrictions that only passively corrupted parties can be crashed by the adversary, and semi-malicious corruption is not tolerated. Since leaking a small amount of information is unavoidable, as is the need to abort the protocol in case of failures, our protocols seem to achieve the best possible goal in a model with fail-corruption.

Further contributions of the paper are applications of the protocol to obtain secure MPC protocols, which requires a way to bound the aggregated leakage when multiple small-leakage protocols are executed in parallel or sequentially. Moreover, while previous protocols are based on the DDH assumption, a new so-called PKCR public-key encryption scheme based on the LWE assumption is proposed, allowing to base topology-hiding computation on LWE. Furthermore, a protocol using fully-homomorphic encryption achieving very low round complexity is proposed.

1 Introduction

1.1 Topology-Hiding Computation

Secure communication over an insecure network is one of the fundamental goals of cryptography. The security goal can be to hide different aspects of the communication, ranging from the content (secrecy), the participants' identity (anonymity), the existence of communication (steganography), to hiding the topology of the underlying network in case it is not complete.

Incomplete networks arise in many contexts, such as the Internet of Things (IoT) or ad-hoc vehicular networks. Hiding the topology can, for example, be important because the position of a node within the network depends on the node's location. This could in information about the

⋆ This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 1122374. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors(s) and do not necessarily reflect the views of the National Science Foundation. Research also supported in part by NSF Grants CNS-1350619 and CNS-1414119, and by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Research Office under contracts W911NF-15-C-0226 and W911NF-15-C-0236.

⋆⋆ Supported in part by ISF grant no. 1790/13 and by the Bar-Ilan Cyber-center.

⋆⋆⋆ Research was supported by the Zurich Information Security and Privacy Center (ZISC).

† Work partly done while author was at ETH Zurich. Author was supported by advanced ERC grant MPCPRO.

node's identity or other confidential parameters. The goal is that parties, and even colluding sets of parties, can not learn anything about the network, except their immediate neighbors.

Incomplete networks have been studied in the context of communication security, referred to as secure message transmission (see, e.g.[DDWY90]), where the goal is to enable communication between any pair of entities, despite an incomplete communication graph. Also, anonymous communication has been studied extensively (see, e.g. [Cha81, RC88, SGR97]). Here, the goal is to hide the identity of the sender and receiver in a message transmission. A classical technique to achieve anonymity is the so-called mix-net technique, introduced by Chaum [Cha81]. Here, *mix* servers are used as proxies which shuffle messages sent between peers to disable an eavesdropper from following a message's path. The onion routing technique [SGR97, RC88] is perhaps the most known instantiation of the mix-technique. Another anonymity technique known as *Dining Cryptographers networks*, in short DC-nets, was introduced in [Cha88] (see also [Bd90, GJ04]). However, none of these approaches can be used to hide the network topology. In fact, message transmission protocols assume (for their execution) that the network graph is public knowledge.

The problem of *topology-hiding communication* was introduced by Moran et al. [MOR15]. The authors propose a broadcast protocol in the cryptographic setting, which does not reveal any additional information about the network topology to an adversary who can access the internal state of any number of passively corrupted parties (that is, they consider the semi-honest setting). This allows to achieve topology-hiding MPC using standard techniques to transform broadcast channels into secure point-to-point channels. At a very high level, [MOR15] uses a series of nested multi-party computations, in which each node is emulated by a secure computation of its neighbor. This emulation then extends to the entire graph recursively. In [HMTZ16], the authors improve this result and provide a construction that makes only black-box use of encryption and where the security is based on the DDH assumption. However, both results are feasible only for graphs with logarithmic diameter. Topology hiding communication for certain classes of graphs with large diameter was described in [AM17]. This result was finally extended to allow for arbitrary (connected) graphs in [ALM17a].

A natural next step is to extend these results to settings with more powerful adversaries. Unfortunately, even a protocol in the setting with fail-corruptions (in addition to passive corruptions) turns out to be difficult to achieve. In fact, as shown already in [MOR15], some leakage in the fail-stop setting is inherent. It is therefore no surprise that all previous protocols (secure against passive corruptions) leak information about the network topology if the adversary can crash parties. The core problem is that crashes can interrupt the communication flow of the protocol at any point and at any time. If not properly dealt with by the protocol, those outages cause shock waves of miscommunication, which allows the adversary to probe the network topology.

A first step in this direction was recently achieved in [BBMM18] where a protocol for topology-hiding communication secure against a fail-stop adversary is given. However, the resilience against crashes comes at a hefty price; the protocol requires that parties have access to secure hardware modules which are initialized with correlated, pre-shared keys. Their protocol provides security with abort and the leakage is arbitrarily small.

In the information-theoretic setting, the main result is negative [HJ07]: any MPC protocol in the information-theoretic setting inherently leaks information about the network graph. They also show that if the routing table is leaked, one can construct an MPC protocol which leaks no additional information.

1.2 Comparison to Previous Work

In [ALM17a] the authors present a broadcast protocol for the semi-honest setting based on random walks. This broadcast protocol is then compiled into a full topology-hiding computation protocol. However, the random walk protocol fails spectacularly in the presence of fail-stop adversaries, leaking a lot of information about the structure of the graph. Every time a node

aborts, any number of walks get cut, meaning that they no longer carry any information. When this happens, adversarial nodes get to see which walks fail along which edges, and can get a good idea of where the aborting nodes are in the graph.

We also note that, while we use ideas from [BBMM18], which achieves the desired result in a trusted-hardware model, we cannot simply use their protocol and substitute the secure hardware box for a standard primitive. In particular, they use the fact that each node can maintain an encrypted "image" of the entire graph by combining information from all neighbors, and use that information to decide whether to give output or abort. This appears to require both some form of obfuscation and a trusted setup, whereas our protocol uses neither.

1.3 Contributions

In this paper we propose the first topology-hiding MPC protocol secure against passive and fail-stop adversaries (with arbitrarily small leakage) that is based on standard assumptions. Our protocol does not require setup, and its security can be based on either the DDH, QR or LWE assumptions. A comparison of our results to previous works in topology-hiding communication is found in Table 1.

Theorem 1 (informal). *If DDH, QR or LWE is hard, then for any MPC functionality $\mathcal{F}$, there exists a topology-hiding protocol realizing $\mathcal{F}$ for any network graph G leaking at most an arbitrarily small fraction p of a bit, which is secure against an adversary that does any number of static passive corruptions and adaptive crashes. The round and communication complexity is polynomial in the security parameter κ and $1/p$.*

Table 1. Adversarial model and security assumptions of existing topology-hiding broadcast protocols. The table also shows the class of graphs for which the protocols have polynomial communication complexity in the security parameter and the number of parties.

Adversary	Graph	Hardness Asm.	Model	Reference
semi-honest	log diam.	Trapdoor Perm.	Standard	[MOR15]
	log diam.	DDH	Standard	[HMTZ16]
	cycles, trees, log circum.	DDH	Standard	[AM17]
	arbitrary	DDH or QR	Standard	[ALM17a]
fail-stop	arbitrary	OWF	Trusted Hardware	[BBMM18]
semi-malicious & fail-stop	arbitrary	DDH or QR or LWE	Standard	[This work]

Our topology-hiding MPC protocol is obtained by compiling a MPC protocol from a topology-hiding broadcast protocol leaking at most a fraction p of a bit. We note that although it is well known that without leakage any functionality can be implemented on top of secure communication, this statement cannot be directly lifted to the setting with leakage. In essence, if a communication protocol is used multiple times, it leaks multiple bits. However, we show that our broadcast protocol, leaking at most a fraction p of a bit, can be executed sequentially and in parallel, such that the result leaks also at most the same fraction p. As a consequence, any protocol can be compiled into one that hides topology and known results on implementing any multiparty computation can be lifted to the topology hiding setting. However, this incurs a multiplicative overhead in the round complexity.

We then present a topology hiding protocol to evaluate any poly-time function using FHE whose round complexity will amount to that of a single broadcast execution. To do that, we first define an enhanced encryption scheme, which we call *Deeply Fully-Homomorphic Public-Key Encryption* (DFH-PKE), with similar properties as the PKCR scheme presented in [AM17,

ALM17a] and provide an instantiation of DFH-PKE under FHE. Next, we show how to obtain a protocol using DFH-PKE to evaluate any poly-time function in a topology hiding manner.

We also explore another natural extension of semi-honest corruption, the so-called *semi-malicious* setting. As for passive corruption, the adversary selects a set of parties and gets access to their internal state. But in addition, the adversary can also set their randomness during the protocol execution. This models the setting where a party uses an untrusted source of randomness which could be under the control of the adversary. This scenario is of interest as tampered randomness sources have caused many security breaches in the past [HDWH12, CNE+14]. In this paper, we propose a general compiler that enhances the security of protocols that tolerate passive corruption with crashes to semi-malicious corruption with crashes.

2 Preliminaries

2.1 Notation

For a public-key $\mathtt{pk}$ and a message m, we denote the encryption of m under $\mathtt{pk}$ by $[m]_{\mathtt{pk}}$. Furthermore, for k messages $m_1, \ldots, m_k$, we denote by $[m_1, \ldots, m_k]_{\mathtt{pk}}$ a vector, containing the k encryptions of messages m_i under the same key $\mathtt{pk}$.

For an algorithm $\mathsf{A}(\cdot)$, we write $\mathsf{A}(\cdot\,; U^*)$ whenever the randomness used in $\mathsf{A}(\cdot)$ should be made explicit and comes from a uniform distribution. By $\approx_c$ we denote that two distribution ensembles are computationally indistinguishable.

2.2 Model of Topology-Hiding Communication

Adversary. Most of our results concern an adversary, who can *statically passively corrupt* an arbitrary set of parties $\mathcal{Z}^p$, with $|\mathcal{Z}^p| < n$. Passively corrupted parties follow the protocol instructions (this includes the generation of randomness), but the adversary can access their internal state during the protocol.

A *semi-malicious* corruption (see, e.g., [AJL+12]) is a stronger variant of a passive corruption. Again, we assume that the adversary selects any set of semi-malicious parties $\mathcal{Z}^s$ with $|\mathcal{Z}^s| < n$ before the protocol execution. These parties follow the protocol instructions, but the adversary can access their internal state and can additionally choose their randomness.

A *fail-stop* adversary can adaptively crash parties. After being crashed, a party stops sending messages. Note that crashed parties are not necessarily corrupted. In particular, the adversary has no access to the internal state of a crashed party unless it is in the set of corrupted parties. This type of fail-stop adversary is stronger and more general than the one used in [BBMM18], where only passively corrupted parties can be crashed. In particular, in our model the adversary does not necessarily learn the neighbors of crashed parties, whereas in [BBMM18] they are revealed to it by definition.

Communication Model. We state our results in the UC framework. We consider a synchronous communication network. Following the approach in [MOR15], to model the restricted communication network we define the $\mathcal{F}_{\text{NET}}$-hybrid model. The $\mathcal{F}_{\text{NET}}$ functionality takes as input a description of the graph network from a special "graph party" P_{graph} and then returns to each party P_i a description of its neighborhood. After that, the functionality acts as an "ideal channel" that allows parties to communicate with their neighbors according to the graph network.

Similarly to [BBMM18], we change the $\mathcal{F}_{\text{NET}}$ functionality from [MOR15] to deal with a fail-stop adversary.

Functionality $\mathcal{F}_{\text{NET}}$

The functionality keeps the following variables: the set of crashed parties $\mathcal{C}$ and the graph G. Initially, $\mathcal{C} = \varnothing$ and $G = (\varnothing, \varnothing)$.

Initialization Step:

1: The party P_{graph} sends graph G' to $\mathcal{F}_{\text{NET}}$. $\mathcal{F}_{\text{NET}}$ sets $G = G'$.
2: $\mathcal{F}_{\text{NET}}$ sends to each party P_i its neighborhood $\mathbf{N}_G(P_i)$.

Communication Step:

1: If the adversary crashes party P_i, then $\mathcal{F}_{\text{NET}}$ sets $\mathcal{C} = \mathcal{C} \cup \{P_i\}$.
2: If a party P_i sends the command (SEND, j, m), where $P_j \in \mathbf{N}_G(P_i)$ and m is the message to P_j, to $\mathcal{F}_{\text{NET}}$ and $P_i \notin \mathcal{C}$, then $\mathcal{F}_{\text{NET}}$ outputs (i, m) to party P_j.

Observe that since $\mathcal{F}_{\text{NET}}$ gives local information about the network graph to all corrupted parties, any ideal-world adversary should also have access to this information. For this reason, similar to [MOR15], we use in the ideal-world the functionality $\mathcal{F}_{\text{INFO}}$, which contains only the Initialization Step of $\mathcal{F}_{\text{NET}}$.

To model leakage we extend $\mathcal{F}_{\text{INFO}}$ by a leakage phase, where the adversary can query a (possibly probabilistic) leakage function $\mathcal{L}$ once. The inputs to $\mathcal{L}$ include the network graph, the set of crashed parties and arbitrary input from the adversary.

We say that a protocol leaks one bit of information if the leakage function $\mathcal{L}$ outputs one bit. We also consider the notion of leaking a fraction p of a bit. This is modeled by having $\mathcal{L}$ output the bit only with probability p (otherwise, $\mathcal{L}$ outputs a special symbol $\perp$). Here our model differs from the one in [BBMM18], where in case of the fractional leakage, $\mathcal{L}$ always gives the output, but the simulator is restricted to query its oracle with probability p over its randomness. As noted there, the formulation we use is stronger. We denote by $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ the information functionality with leakage function $\mathcal{L}$.

Functionality $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$

The functionality keeps the following variables: the set of crashed parties $\mathcal{C}$ and the graph G. Initially, $\mathcal{C} = \varnothing$ and $G = (\varnothing, \varnothing)$.

Initialization Step:

1: The party P_{graph} sends graph $G' = (V, E)$ to $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$. $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ sets $G = G'$.
2: $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ sends to each party P_i its neighborhood $\mathbf{N}_G(P_i)$.

Leakage Step:

1: If the adversary crashes party P_i, then $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ sets $\mathcal{C} = \mathcal{C} \cup \{P_i\}$.
2: If the adversary sends the command (LEAK, q) to $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ for the first time, then $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ outputs $\mathcal{L}(q, \mathcal{C}, G)$ to the adversary.

Security Model. Our protocols provide security with abort. In particular, the adversary can choose some parties, who do not receive the output (while the others still do). That is, no guaranteed output delivery and no fairness is provided. Moreover, the adversary sees the output before the honest parties and can later decide which of them should receive it.

Technically, we model such ability in the UC framework as follows: First, the ideal world adversary receives from the ideal functionality the outputs of the corrupted parties. Then, it inputs to the functionality an *abort vector* containing a list of parties who do not receive the output.

Definition 1. *We say that a protocol Π topology-hidingly realizes a functionality $\mathcal{F}$ with $\mathcal{L}$-leakage, in the presence of an adversary who can statically passive corrupt and adaptively crash any number of parties, if it UC-realizes $(\mathcal{F}_{\text{INFO}}^{\mathcal{L}} \parallel \mathcal{F})$ in the $\mathcal{F}_{\text{NET}}$-hybrid model.*

2.3 Background

Graphs and Random Walks. In an undirected graph $G = (V, E)$ we denote by $\mathbf{N}_G(P_i)$ the neighborhood of $P_i \in V$. The k-neighborhood of a party $P_i \in V$ is the set of all parties in V within distance k to P_i.

In our work we use the following lemma from [ALM17a]. It states that in an undirected connected graph G, the probability that a random walk of length $8|V|^3\tau$ covers G is at least $1 - \frac{1}{2^\tau}$.

Lemma 1 ([ALM17a]). *Let $G = (V, E)$ be an undirected connected graph. Further let $\mathcal{W}(u, \tau)$ be a random variable whose value is the set of nodes covered by a random walk starting from u and taking $8|V|^3\tau$ steps. We have*

$$\Pr_{\mathcal{W}}[\mathcal{W}(u, \tau) = V] \geq 1 - \frac{1}{2^\tau}.$$

PKCR Encryption. As in [ALM17a], our protocols require a public key encryption scheme with additional properties, called *Privately Key Commutative and Rerandomizable encryption.* We assume that the message space is bits. Then, a PKCR encryption scheme should be: privately key commutative and homomorphic with respect to the OR operation[5]. We formally define these properties below.

Let $\mathcal{PK}$, $\mathcal{SK}$ and $\mathcal{C}$ denote the public key, secret key and ciphertext spaces. As any public key encryption scheme, a PKCR scheme contains the algorithms $\mathsf{KeyGen} : \{0,1\}^* \to \mathcal{PK} \times \mathcal{SK}$, $\mathsf{Encrypt} : \{0,1\} \times \mathcal{PK} \to \mathcal{C}$ and $\mathsf{Decrypt} : \mathcal{C} \times \mathcal{SK} \to \{0,1\}$ for key generation, encryption and decryption respectively (where KeyGen takes as input the security parameter).

Privately Key-Commutative. We require $\mathcal{PK}$ to form a commutative group under the operation $\circledast$. So, given any $\mathtt{pk}_1, \mathtt{pk}_2 \in \mathcal{PK}$, we can efficiently compute $\mathtt{pk}_3 = \mathtt{pk}_1 \circledast \mathtt{pk}_2 \in \mathcal{PK}$ and for every $\mathtt{pk}$, there exists an inverse denoted $\mathtt{pk}^{-1}$.

This group must interact well with ciphertexts; there exists a pair of efficiently computable algorithms $\mathsf{AddLayer} : \mathcal{C} \times \mathcal{SK} \to \mathcal{C}$ and $\mathsf{DelLayer} : \mathcal{C} \times \mathcal{SK} \to \mathcal{C}$ such that

- For every public key pair $\mathtt{pk}_1, \mathtt{pk}_2 \in \mathcal{PK}$ with corresponding secret keys $\mathtt{sk}_1$ and $\mathtt{sk}_2$, message $m \in \mathcal{M}$, and ciphertext $c = [m]_{\mathtt{pk}_1}$,
$$\mathsf{AddLayer}(c, \mathtt{sk}_2) = [m]_{\mathtt{pk}_1 \circledast \mathtt{pk}_2}.$$
- For every public key pair $\mathtt{pk}_1, \mathtt{pk}_2 \in \mathcal{PK}$ with corresponding secret keys $\mathtt{sk}_1$ and $\mathtt{sk}_2$, message $m \in \mathcal{M}$, and ciphertext $c = [m]_{\mathtt{pk}_1}$,
$$\mathsf{DelLayer}(c, \mathtt{sk}_2) = [m]_{\mathtt{pk}_1 \circledast \mathtt{pk}_2^{-1}}.$$

Notice that we need the secret key to perform these operations, hence the property is called *privately* key-commutative.

OR-Homomorphic. We also require the encryption scheme to be OR-homomorphic, but in such a way that parties cannot tell how many 1's or 0's were OR'd (or who OR'd them). We need an efficiently-evaluatable homomorphic-OR algorithm, $\mathsf{HomOR} : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, to satisfy the following: for every two messages $m, m' \in \{0,1\}$ and every two ciphertexts $c, c' \in \mathcal{C}$ such that $\mathsf{Decrypt}(c, \mathtt{sk}) = m$ and $\mathsf{Decrypt}(c, \mathtt{sk}) = m'$,

$$\{(m, m', c, c', \mathtt{pk}, \mathsf{Encrypt}(m \vee m', \mathtt{pk}; U^*))\}$$
$$\approx_c$$
$$\{(m, m', c, c', \mathtt{pk}, \mathsf{HomOR}(c, c', \mathtt{pk}; U^*))\}$$

[5] PKCR encryption was introduced in [AM17, ALM17a], where it had three additional properties: key commutativity, homomorphism and rerandomization, hence, it was called Privately Key Commutative and *Rerandomizable* encryption. However, rerandomization is actually implied by the strengthened notion of homomorphism. Therefore, we decided to not include the property, but keep the name.

Note that this is a stronger definition for homomorphism than usual; usually we only require correctness, not computational indistinguishability.

In [HMTZ16], [AM17] and [ALM17a], the authors discuss how to get this kind of homomorphic OR under the DDH assumption, and later [ALM17b] show how to get it with the QR assumption. For more details on other kinds of homomorphic cryptosystems that can be compiled into OR-homomorphic cryptosystems, see [ALM17b].

Random Walk Approach [ALM17a]. Our protocol builds upon the protocol from [ALM17a]. We give a high level overview. To achieve broadcast, the protocol computes the OR. Every party has an input bit: the sender inputs the broadcast bit and all other parties use 0 as input bit. Computing the OR of all those bits is thus equivalent to broadcasting the sender's message.

First, let us explain a simplified version of the protocol that is unfortunately not sound, but gets the basic principal across. Each node encrypts its bit under a public key and forwards it to a random neighbor. The neighbor OR's its own bit, adds a fresh public key layer, and it forwards the ciphertext to a randomly chosen neighbor. Eventually, after about $O(\kappa n^3)$ steps, the random walk of every message visits every node in the graph, and therefore, every message will contain the OR of all bits in the network. Now we start the backwards phase, reversing the walk and peeling off layers of encryption.

This scheme is not sound because seeing where the random walks are coming from reveals information about the graph! So, we need to disguise that information. We will do so by using correlated random walks, and will have a walk running down each direction of each edge at each step (so $2\times$ number of edges number of walks total). The walks are correlated, but still random. This way, at each step, each node just sees encrypted messages all under new and different keys from each of its neighbors. So, intuitively, there is no way for a node to tell anything about where a walk came from.

3 Topology-Hiding Broadcast

In this section we present a protocol, which securely realizes the broadcast functionality $\mathcal{F}_{\text{BC}}$ (with abort) in the $\mathcal{F}_{\text{NET}}$-hybrid world and leaks at most an arbitrarily small (but not negligible) fraction of a bit. If no crashes occur, the protocol does not leak any information. The protocol is secure against an adversary that (a) controls an arbitrary static set of passively corrupted parties and (b) adaptively crashes any number of parties. Security can be based either on the DDH, the QR or the LWE assumption. To build intuition we first present the simple protocol variant which leaks at most one bit.

Functionality $\mathcal{F}_{\text{BC}}$

When a party P_i sends a bit $b \in \{0,1\}$ to the functionality $\mathcal{F}_{\text{BC}}$, then $\mathcal{F}_{\text{BC}}$ sends b to each party $P_j \in \mathcal{P}$.

3.1 Protocol Leaking One Bit

We first introduce the broadcast protocol variant BC-OB which leaks at most one-bit. The protocol is divided into n consecutive phases, where, in each phase, the parties execute a modification of the random-walk protocol from [ALM17a]. More specifically, we introduce the following modifications:

Single Output Party: There will be n phases. In each phase only one party, P_o, gets the output. Moreover, it learns the output from exactly one of the random walks it starts.
To implement this, in the respective phase all parties except P_o start their random walks with encryptions of 1 instead of their input bits. This ensures that the outputs they get

from the random walks will always be 1. We call these walks *dummy* since the contain no information. Party P_o, on the other hand, starts exactly one random walk with its actual input bit (the other walks it starts with encryptions of 1). This ensures (in case no party crashes) that P_o actually learns the broadcast bit.

Happiness Indicator: Every party P_i holds an *unhappy-bit* u_i. Initially, every P_i is happy, i.e., $u_i = 0$. If a neighbor of P_i crashes, then in the next phase P_i becomes unhappy and sets $u_i = 1$. The idea is that an unhappy party makes all phases following the crash become dummy.

This is implemented by having the parties send along the random walk, instead of a single bit, an encrypted tuple $[b, u]_{\mathtt{pk}}$. The bit u is the OR of the unhappy-bits of the parties in the walk, while b is the OR of their input bits and their unhappy-bits. In other words, a party P_i on the walk homomorphically ORs $b_i \vee u_i$ to b and u_i to u.

Intuitively, if all parties on the walk were happy at the time of adding their bits, b will actually contain the OR of their input bits and u will be set to 0. On the other hand, if any party was unhappy, b will always be set to 1, and $u = 1$ will indicate an abort.

Intuitively, the adversary learns a bit of information only if it manages to break the one random walk which P_o started with its input bit (all other walks contain the tuple $[1, 1]$). Moreover, if it crashes a party, then all phases following the one with the crash abort, hence, they do not leak any information.

More formally, parties execute, in each phase, protocol RandomWalkPhase. This protocol takes as global inputs the length $\mathtt{T}$ of the random walk and the P_o which should get output. Additionally, each party P_i has input (d_i, b_i, u_i) where d_i is its number of neighbors, u_i is its unhappy-bit, and b_i is its input bit.

Protocol $\mathsf{RandomWalkPhase}(\mathtt{T}, P_o, (d_i, b_i, u_i)_{P_i \in \mathcal{P}})$

Initialization Stage:

1: Each party P_i generates $\mathtt{T} \cdot d_i$ keypairs $(\mathtt{pk}_{i\to j}^{(r)}, \mathtt{sk}_{i\to j}^{(r)}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ where $r \in \{1, \ldots, \mathtt{T}\}$ and $j \in \{1, \ldots, d_i\}$.

2: Each party P_i generates $\mathtt{T} - 1$ random permutations on d_i elements $\left\{\pi_i^{(2)}, \ldots, \pi_i^{(\mathtt{T})}\right\}$

3: For each party P_i, if any of P_i's neighbors crashed in any phase before the current one, then P_i becomes unhappy, i.e., sets $u_i = 1$.

Aggregate Stage: Each party P_i does the following:

1: **if** P_i is the recipient P_o **then**

2: Party P_i sends to the first neighbor the ciphertext $[b_i \vee u_i, u_i]_{\mathtt{pk}_{i\to 1}^{(1)}}$ and the public key $\mathtt{pk}_{i\to 1}^{(1)}$, and to any other neighbor P_j it sends ciphertext $[1, 1]_{\mathtt{pk}_{i\to j}^{(1)}}$ and the public key $\mathtt{pk}_{i\to j}^{(1)}$.

3: **else**

4: Party P_i sends to each neighbor P_j ciphertext $[1, 1]_{\mathtt{pk}_{i\to j}^{(1)}}$ and the key $\mathtt{pk}_{i\to j}^{(1)}$.

5: **end if**

6: // Add layer while ORing own input bit

7: **for** any round r from 2 to $\mathtt{T}$ **do**

8: For each neighbor P_j of P_i, do the following (let $k = \pi_i^{(r)}(j)$):

9: **if** P_i did not receive a message from P_j **then**

10: Party P_i sends ciphertext $[1, 1]_{\mathtt{pk}_{i\to k}^{(r)}}$ and key $\mathtt{pk}_{i\to k}^{(r)}$ to neighbor P_k.

11: **else** // AddLayer and HomOR are applied component-wise

12: Let $\mathtt{c}_{j\to i}^{(r-1)}$ and $\overline{\mathtt{pk}}_{j\to i}^{(r-1)}$ be the ciphertext and the public key P_i received from P_j. Party P_i computes $\overline{\mathtt{pk}}_{i\to k}^{(r)} = \overline{\mathtt{pk}}_{j\to i}^{(r-1)} \circledast \mathtt{pk}_{i\to k}^{(r)}$ and $\hat{\mathtt{c}}_{i\to k}^{(r)} \leftarrow \mathsf{AddLayer}\left(\mathtt{c}_{j\to i}^{(r-1)}, \mathtt{sk}_{i\to k}^{(r)}\right)$.

13: P_i computes $[b_i \vee u_i, u_i]_{\overline{\mathtt{pk}}_{i\to k}^{(r)}}$ and $\mathtt{c}_{i\to k}^{(r)} = \mathsf{HomOR}\left([b_i \vee u_i, u_i]_{\overline{\mathtt{pk}}_{i\to k}^{(r)}}, \hat{\mathtt{c}}_{i\to k}^{(r)}, \overline{\mathtt{pk}}_{i\to k}^{(r)}\right)$.

14: Party P_i sends ciphertext $\mathsf{c}_{i\to k}^{(r)}$ and public key $\overline{\mathtt{pk}}_{i\to k}^{(r)}$ to neighbor P_k.
15: **end if**
16: **end for**

Decrypt Stage: Each party P_i does the following:

1: For each neighbor P_j of P_i, if P_i did not receive a message from P_j at round T of the Aggregate Stage, then it sends ciphertext $\mathsf{e}_{i\to j}^{(\mathtt{T})} = [1,1]_{\overline{\mathtt{pk}}_{j\to i}^{(\mathtt{T})}}$ to P_j. Otherwise, P_i sends to P_j $\mathsf{e}_{i\to j}^{(\mathtt{T})} = \mathsf{HomOR}\left([b_i \vee u_i, u_i]_{\overline{\mathtt{pk}}_{j\to i}^{(\mathtt{T})}}, \mathsf{c}_{j\to i}^{(\mathtt{T})}, \overline{\mathtt{pk}}_{j\to i}^{(\mathtt{T})}\right)$.
2: **for** any round r from $\mathtt{T}$ to 2 **do**
3: For each neighbor P_k of P_i:
4: **if** P_i did not receive a message from P_k **then**
5: Party P_i sends $\mathsf{e}_{i\to j}^{(r-1)} = [1,1]_{\overline{\mathtt{pk}}_{j\to i}^{(r-1)}}$ to neighbor P_j, where $k = \pi_i^{(r)}(j)$.
6: **else**
7: Denote by $\mathsf{e}_{k\to i}^{(r)}$ the ciphertext P_i received from P_k, where $k = \pi_i^{(r)}(j)$. Party P_i sends $\mathsf{e}_{i\to j}^{(r-1)} = \mathsf{DelLayer}\left(\mathsf{e}_{k\to i}^{(r)}, \mathtt{sk}_{i\to k}^{(r)}\right)$ to neighbor P_j.
8: **end if**
9: **end for**
10: If P_i is the recipient P_o, then it computes $(b,u) = \mathsf{Decrypt}(\mathsf{e}_{1\to i}^{(1)}, \mathtt{sk}_{i\to 1}^{(1)})$ and **outputs** (b,u,u_i). Otherwise, it **outputs** $(1,0,u_i)$.

The actual protocol BC-OB consists of n consecutive runs of the random walk phase protocol RandomWalkPhase.

Protocol BC-OB$(\mathtt{T}, (d_i,b_i)_{P_i\in\mathcal{P}})$

Each party P_i keeps bits b_i^{out}, u_i^{out} and u_i, and sets $u_i = 0$.
for o from 1 to n **do**
Parties jointly execute
$\left((b_i^{tmp}, v_i^{tmp}, u_i^{tmp})_{P_i\in\mathcal{P}}\right) = \mathsf{RandomWalkPhase}(\mathtt{T}, P_o, (d_i,b_i,u_i)_{P_i\in\mathcal{P}})$.
Each party P_i sets $u_i = u_i^{tmp}$.
Party P_o sets $b_o^{out} = b_o^{tmp}$, $u_o^{out} = v_o^{tmp}$.
end for
For each party P_i, **if** $u_i^{out} = 0$ **then** party P_i **outputs** b_i^{out}.

The protocol BC-OB leaks information about the topology of the graph during the execution of RandomWalkPhase, in which the first crash occurs. (Every execution before the first crash proceeds almost exactly as the protocol in [ALM17a] and in every execution afterwards all values are blinded by the unhappy-bit u.) We model the leaked information by a query to the leakage function $\mathcal{L}_{OB}$. The function outputs only one bit and, since the functionality $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ allows for only one query to the leakage function, the protocol leaks overall one bit of information.

The inputs passed to $\mathcal{L}_{OB}$ are: the graph G and the set $\mathcal{C}$ of crashed parties, passed to the function by $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$, and a triple $(F, P_s, \mathtt{T}')$, passed by the simulator. The idea is that the simulator needs to know whether the walk carrying the output succeeded or not, and this depends on the graph G. More precisely, the set F contains a list of pairs (P_f, r), where r is the number of rounds in the execution of RandomWalkPhase, at which P_f crashed. $\mathcal{L}_{OB}$ tells the simulator whether any of the crashes in F disconnected a freshly generated random walk of length $\mathtt{T}'$, starting at given party P_s.

Function $\mathcal{L}_{OB}((F,P_s,\mathtt{T}'),\mathcal{C},G)$

if for any $(P_f,r)\in F$, $P_f\notin\mathcal{C}$ **then** Return 0.
else
Generate in G a random walk of length $\mathtt{T}'$ starting at P_s.

Return 1 if for any $(P_f, r) \in F$ removing party P_f after r rounds disconnects the walk and 0 otherwise.
end if

We prove the following theorem in Section A.1.

Theorem 2. *For κ security parameter and $\mathsf{T} = 8n^3(\log(n)+\kappa)$ protocol $\mathsf{BC\text{-}OB}(\mathsf{T}, (d_i, b_i)_{P_i \in \mathcal{P}})$ topology-hidingly realizes $\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{OB}} || \mathcal{F}_{\text{BC}}$ (with abort) in the $\mathcal{F}_{\text{NET}}$ hybrid-world, where the leakage function $\mathcal{L}_{OB}$ is the one defined as above. If no crashes occur, then there is no abort and there is no leakage.*

3.2 Protocol Leaking a Fraction of a Bit

We now show how to go from BC-OB to the actual broadcast protocol $\mathsf{BC\text{-}FB}_p$ which leaks only a fraction p of a bit. The leakage parameter p can be arbitrarily small. However, the complexity of the protocol is proportional to $1/p$. As a consequence, $1/p$ must be polynomial and p cannot be negligible.

The idea is to leverage the fact that the adversary can gain information in only one execution of RandomWalkPhase. Imagine that RandomWalkPhase succeeds only with a small probability p, and otherwise the output bit b is 1. Moreover, assume that during RandomWalkPhase the adversary does not learn whether it will fail until it can decrypt the output.

We can now, for each phase, repeat RandomWalkPhase ρ times, so that with overwhelming probability one of the repetitions does not fail. A party P_o can then compute its output as the AND of outputs from all repetitions (or abort if any repetition aborted). On the other hand, the adversary can choose only one execution of RandomWalkPhase, in which it learns one bit of information (all subsequent repetitions will abort). Moreover, it must choose it before it knows whether the execution succeeds. Hence, the adversary learns one bit of information only with probability p.

What is left is to modify RandomWalkPhase, so that it succeeds only with probability p, and so that the adversary does not know whether it will succeed. We only change the Aggregate Stage. Instead of an encrypted tuple $[b, u]$, the parties send along the walk $\lfloor 1/p \rfloor + 1$ encrypted bits $[b^1, \ldots, b^{\lfloor 1/p \rfloor}, u]$, where u again is the OR of the unhappy-bits, and every b^k is a copy the bit b in RandomWalkPhase, with some caveats. For each phase o, and for every party $P_i \neq P_o$, all b^k are copies of b in the walk and they all contain 1. For P_o, only one of the bits, b^k, contains the OR, while the rest is initially set to 1.

During the Aggregate Stage, the parties process every ciphertext corresponding to a bit b^k the same way they processed the encryption of b in the
RandomWalkPhase. Then, before sending the ciphertexts to the next party on the walk, the encryptions of the bits b^k are randomly shuffled. (This way, as long as the walk traverses an honest party, the adversary does not know which of the ciphertexts contain dummy values.) At the end of the Aggregate Stage (after $\mathtt{T}$ rounds), the last party chooses uniformly at random one of the $\lfloor 1/p \rfloor$ ciphertexts and uses it, together with the encryption of the unhappy-bit, to execute the Decrypt Stage as in RandomWalkPhase.

The information leaked by $\mathsf{BC\text{-}FB}_p$ is modeled by the following function $\mathcal{L}_{FB_p}$.

Function $\mathcal{L}_{FB_p}((F, P_s, \mathtt{T}'), \mathcal{C}, G)$

Let $p' = 1/\lfloor 1/p \rfloor$. With probability p', return $\mathcal{L}_{OB}((F, P_s, \mathtt{T}'), \mathcal{C}, G)$ and with probability $1 - p'$ return $\perp$.

A formal description of the modified protocol $\mathsf{ProbabilisticRandomWalkPhase}_p$ and a proof of the following theorem can be found in Section A.2.

Theorem 3. *Let κ be the security parameter. For $\tau = \log(n) + \kappa$, $\mathsf{T} = 8n^3\tau$, and $\rho = \tau/(p' - 2^{-\tau})$, where $p' = 1/\lfloor 1/p \rfloor$, protocol $\mathsf{BC\text{-}FB}_p(\mathsf{T}, \rho, (d_i, b_i)_{P_i \in \mathcal{P}})$ topology-hidingly realizes*

$\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{FB_p}} || \mathcal{F}_{\text{BC}}$ *(with abort) in the* $\mathcal{F}_{\text{NET}}$ *hybrid-world, where the leakage function* $\mathcal{L}_{FB_p}$ *is the one defined as above. If no crashes occur, then there is no abort and there is no leakage.*

4 From Broadcast to Topology-Hiding Computation

We showed how to get topology-hiding broadcasts. To get additional functionality (e.g. for compiling MPC protocols), we have to be able to compose these broadcasts. When there is no leakage, this is straightforward: we can run as many broadcasts in parallel or in sequence as we want and they will not affect each other. However, if we consider a broadcast secure in the fail-stop model that leaks at most 1 bit, composing t of these broadcasts could lead to leaking t bits.

The first step towards implementing any functionality in a topology-hiding way is to modify our broadcast protocol to a topology-hiding all-to-all multibit broadcast, without aggregating leakage. Then, we show how to sequentially compose such broadcasts, again without adding leakage. Finally, one can use standard techniques to compile MPC protocols from broadcast. In the following, we give a high level overview of each step. A detailed description of the transformations can be found in Section B.

All-to-all Multibit Broadcast. The first observation is that a modification of $\mathsf{BC\text{-}FB}_p$ allows one party to broadcast multiple bits. Instead of sending a single bit b during the random-walk protocol, each party sends a vector $\vec{b}$ of bits encrypted separately under the same key. That is, in each round of the Aggregate Phase, each party sends a vector $[\vec{b_1}, \ldots, \vec{b_\ell}, u]$.

We can extend this protocol to all-to-all multibit broadcast, where each party P_i broadcasts a message $(b_1, \ldots, b_k)$, as follows. Each of the vectors $\vec{b_i}$ in $[\vec{b_1}, \ldots, \vec{b_\ell}, u]$ contains nk bits, and P_i uses the bits from $n(i-1)$ to ni to communicate its message. That is, in the Aggregate Stage, *every* P_i homomorphically OR's $\vec{b_i} = (0, \ldots, 0, b_1, \ldots, b_k, 0, \ldots, 0)$ with the received encrypted vectors.

Sequential execution. All-to-all broadcasts can be composed sequentially by preserving the state of unhappy bits between sequential executions. That is, once some party sees a crash, it will cause all subsequent executions to abort.

Topology-Hiding computation. With the above statements, we conclude that any MPC protocol can be compiled into one that leaks only a fraction p of a bit in total. This is achieved using a public key infrastructure, where in the first round the parties use the topology hiding all-to-all broadcast to send each public key to every other party, and then each round of the MPC protocol is simulated with an all-to-all multibit topology-hiding broadcast. As a corollary, any functionality $\mathcal{F}$ can be implemented by a topology-hiding protocol leaking any fraction p of a bit.

5 Efficient Topology-Hiding Computation with FHE

One thing to note is that compiling MPC from broadcast is rather expensive, especially in the fail-stop model; we need a broadcast for every round. However, we will show that an FHE scheme with additive overhead can be used to evaluate any polynomial-time function f in a topology-hiding manner. Additive overhead applies to ciphertext versus plaintext sizes and to error with respect to all homomorphic operations if necessary. We will employ an altered random walk protocol, and the total number of rounds in this protocol will amount to that of a single broadcast. We remark that FHE with additive overhead can be obtained from subexponential iO and subexponentially secure OWFs (probabilistic iO), as shown in [CLTV15].

5.1 Deeply-Fully-Homomorphic Public-Key Encryption

In the altered random walk protocol, the PKCR scheme is replaced by a deeply-fully-homomorphic PKE scheme (DFH-PKE). Similarly to PKCR, a DFH-PKE scheme is a public-key encryption scheme enhanced by algorithms for adding and deleting layers. However, we do not require that public keys form a group, and we allow the ciphertexts and public keys on different levels (that is, for which a layer has been added a different number of times) to be distinguishable. Moreover, DFH-PKE offers full homomorphism.

This is captured by three additional algorithms: $\mathsf{AddLayer}_r$, $\mathsf{DelLayer}_r$, and HomOp_r, operating on ciphertexts with r layers of encryption (we will call such ciphertexts level-r ciphertexts). A level-r ciphertext is encrypted under a level-r public key (each level can have different key space).

Adding a layer requires a new secret key $\mathtt{sk}$. The algorithm $\mathsf{AddLayer}_r$ takes as input a vector of level-r ciphertexts $[\![\vec{m}]\!]_{\mathbf{pk}}$ encrypted under a level-r public key, the corresponding level-r public key $\mathbf{pk}$, and a new secret key $\mathtt{sk}$. It outputs a vector of level-$(r+1)$ ciphertexts and the level-$(r+1)$ public key, under which it is encrypted. Deleting a layer is the opposite of adding a layer.

With HomOp_r, one can compute any function on a vector of encrypted messages. It takes a vector of level-r ciphertexts encrypted under a level-r public key, the corresponding level-r public key $\mathbf{pk}$ and a function from a permitted set $\mathcal{F}$ of functions. It outputs a level-r ciphertext that contains the output of the function applied to the encrypted messages.

Intuitively, a DFH-PKE scheme is secure if one can simulate any level-r ciphertext without knowing the history of adding and deleting layers. This is captured by the existence of an algorithm $\mathsf{Leveled\text{-}Encrypt}_r$, which takes as input a plain message and a level-r public key, and outputs a level-r ciphertext. We require that for any level-r encryption of a message $\vec{m}$, the output of $\mathsf{AddLayer}_r$ on that ciphertext is indistinguishable from the output of $\mathsf{Leveled\text{-}Encrypt}_{r+1}$ on $\vec{m}$ and a (possibly different) level-$(r+1)$ public key. An analogous property is required for $\mathsf{DelLayer}_r$. We will also require that the output of HomOp_r is indistinguishable from a level-r encryption of the output of the functions applied to the messages. We refer to Section C for a formal definition of a DFH-PKE scheme and to Section C.1 for an instantiation from FHE.

Remark. If we relax DFH-PKE and only require homomorphic evaluation of OR, then this relaxation is implied by any OR-homomorphic PKCR scheme (in PKCR, additionally, all levels of key and ciphertext spaces are the same, and the public key space forms a group). Such OR-homomorphic DFH-PKE would be sufficient to prove the security of the protocols $\mathsf{BC\text{-}OB}$ and $\mathsf{BC\text{-}FB}_p$. However, for simplicity and clarity, we decided to describe our protocols $\mathsf{BC\text{-}OB}$ and $\mathsf{BC\text{-}FB}_p$ from a OR-homomorphic PKCR scheme.

5.2 Topology-Hiding Computation from DFH-PKE

To evaluate any function f, we modify the topology-hiding broadcast protocol (with PKCR replaced by DFH-PKE) in the following way. During the Aggregate Stage, instead of one bit for the OR of all inputs, the parties send a vector of encrypted inputs. At each round, each party homomorphically adds its input together with its id to the vector. The last party on the walk homomorphically evaluates f on the encrypted inputs, and (homomorphically) selects the output of the party who receives it in the current phase. The Decrypt Stage is started with this encrypted result.

Note that we still need a way to make a random walk dummy (this was achieved in $\mathsf{BC\text{-}OB}$ and $\mathsf{BC\text{-}FB}_p$ by starting it with a 1). Here, we will have an additional input bit for the party who starts a walk. In case this bit is set, when homomorphically evaluating f, we (homomorphically) replace the output of f by a special symbol. We refer to Section D for a detailed description of the protocol and a proof of the following theorem.

Theorem 4. *For security parameter κ, $\tau = \log(n) + \kappa$, $T = 8n^3\tau$, and $\rho = \tau/(p' - 2^{-\tau})$, where $p' = 1/\lfloor 1/p \rfloor$, the protocol $\mathsf{DFH\text{-}THC}(T, \rho, (d_i, \mathsf{input}_i)_{P_i \in \mathcal{P}}))$ topology-hidingly evaluates any poly-time function f, $\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{FB_p}} || f$ in the $\mathcal{F}_{\text{NET}}$ hybrid-world.*

6 Security Against Semi-malicious Adversaries

In this section, we show how to generically compile our protocols to provide in addition security against a semi-malicious adversary. The transformed protocol proceeds in two phases: Randomness Generation and Deterministic Execution. In the first phase, we generate the random tapes for all parties and in the second phase we execute the given protocol with parties using the pre-generated random tapes. The tapes are generated in such a way that the tape of each party P_i is the sum of random values generated from each party. Hence, as long as one party is honest, the generated tape is random.

Randomness Generation. The goal of the first phase is to generate for each party P_i a uniform random value r_i, which can then be used as randomness tape of P_i in the phase of Deterministic Execution.[6]

Protocol GenerateRandomness

1: Each party P_i generates $n+1$ uniform random values $s_i^{(0)}, s_i^{(1)}, \ldots, s_i^{(n)}$ and sets $r_i^{(0)} := s_i^{(0)}$.
2: **for** any round r from 1 to n **do**
3: Each party P_i sends $r_i^{(r-1)}$ to all its neighbors.
4: Each party P_i computes $r_i^{(r)}$ as the sum of all values received from its (non-crashed) neighbors in the current round and the value $s_i^{(k)}$.
5: **end for**
6: Each party P_i outputs $r_i := r_i^{(n)}$.

Lemma 2. *Let G' be the network graph without the parties which crashed during the execution of GenerateRandomness. Any party P_i whose connected component in G' contains at least one honest party will output a uniform value r_i. The output of any honest party is not known to the adversary. The protocol GenerateRandomness does not leak any information about the network-graph (even if crashes occur).*

Proof. First observe that all randomness is chosen at the beginning of the first round. The rest of the protocol is completely deterministic. This implies that the adversary has to choose the randomness of corrupted parties independently of the randomness chosen by honest parties.

If party P_i at the end of the protocol execution is in a connected component with honest party P_j, the output r_i is a sum which contains at least one of the values $s_j^{(r)}$ from P_j. That summand is independent of the rest of the summands and uniform random. Thus, r_i is uniform random as well.

Any honest party will (in the last round) compute its output as a sum which contains a locally generated truly random value, which is not known to the adversary. Thus, the output is also not known to the adversary.

Finally, observe that the message pattern seen by a party is determined by its neighborhood. Moreover, the messages received by corrupted parties from honest parties are uniform random values. This implies, that the view of the adversary in this protocol can be easily simulated given the neighborhood of corrupted parties. Thus, the protocol does not leak any information about the network topology. □

[6] To improve overall communication complexity of the protocol the values generated in the first phase could be used as local seeds for a PRG which is then used to generate the actual random tapes.

Transformation to Semi-malicious Security. In the second phase of Deterministic Execution, the parties execute the protocol secure against passive and fail-stop corruptions, but instead of generating fresh randomness during the protocol execution, they use the random tape generated in the first phase.

Protocol EnhanceProtocol(Π)

1: The parties execute GenerateRandomness to generate random tapes.
2: If a party witnessed a crash in GenerateRandomness, it pretends that it witnessed this crash in the first round of the protocol Π.
3: The parties execute Π, using the generated randomness tapes, instead of generating randomness on the fly.

Theorem 5. *Let $\mathcal{F}$ be an MPC functionality and let Π be a protocol that topology-hidingly realizes $\mathcal{F}$ in the presence of static passive corruptions and adaptive crashes. Then, the protocol EnhanceProtocol(Π) topology-hidingly realizes $\mathcal{F}$ in the presence of static semi-malicious corruption and adaptive crashes. The leakage stays the same.*

Proof. (sketch) The randomness generation protocol GenerateRandomness used in the first phase is secure against a semi-malicious fail-stopping adversary. Lemma 2 implies that the random tape of any semi-malicious party that can interact with honest parties is truly uniform random. Moreover, the adversary has no information on the random tapes of honest parties. This implies that the capability of the adversary in the execution of the actual protocol in the second phase (which for fixed random tapes is deterministic) is the same as for an semi-honest fail-stopping adversary. This implies that the leakage of EnhanceProtocol(Π) is the same as for Π as the randomness generation protocol does not leak information (even if crashes occur). □

As a corollary of Theorems 3 and 5, we obtain that any MPC functionality can be realized in a topology-hiding manner secure against an adversary that does any number of static semi-malicious corruptions and adaptive crashes, leaking at most an arbitrary small fraction of information about the topology.

7 LWE based OR-Homomorphic PKCR Encryption

In this section, we show how to get PKCR encryption from the LWE. The basis of our PKCR scheme is the public-key crypto-system proposed in [Reg09].

LWE PKE scheme [Reg09] Let κ be the security parameter of the cryptosystem. The cryptosystem is parameterized by two integers m, q and a probability distribution χ on $\mathbb{Z}_q$. To guarantee security and correctness of the encryption scheme, one can choose $q \geq 2$ to be some prime number between κ^2 and $2\kappa^2$, and let $m = (1+\epsilon)(\kappa+1)\log q$ for some arbitrary constant $\epsilon > 0$. The distribution χ is a discrete gaussian distribution with standard deviation $\alpha(\kappa) := \frac{1}{\sqrt{\kappa} log^2 \kappa}$.

Key Generation: *Setup*: For $i = 1, \ldots, m$, choose m vectors $\mathbf{a}_1, \ldots, \mathbf{a}_m \in \mathbb{Z}_q^\kappa$ independently from the uniform distribution. Let us denote $A \in \mathbb{Z}_q^{m \times \kappa}$ the matrix that contains the vectors $\mathbf{a}_i$ as rows.
Secret Key: Choose $\mathbf{s} \in \mathbb{Z}_q^\kappa$ uniformly at random. The secret key is $\mathtt{sk} = \mathbf{s}$.
Public Key: Choose the error coefficients $e_1, \ldots, e_m \in \mathbb{Z}_q$ independently according to χ. The public key is given by the vectors $b_i = \langle \mathbf{a}_i, \mathtt{sk} \rangle + e_i$. In matrix notation, $\mathtt{pk} = A \cdot \mathtt{sk} + \mathbf{e}$.

Encryption: To encrypt a bit b, we choose uniformly at random $\mathbf{x} \in \{0,1\}^m$. The ciphertext is $c = (\mathbf{x}^\intercal A, \mathbf{x}^\intercal \mathtt{pk} + b\frac{q}{2})$.

Decryption: Given a ciphertext $c = (c_1, c_2)$, the decryption of c is 0 if $c_2 - c_1 \cdot \mathtt{sk}$ is closer to 0 than to $\lfloor \frac{q}{2} \rfloor$ modulo q. Otherwise, the decryption is 1.

Extension to PKCR We now extend the above PKE scheme to satisfy the requirements of a PKCR scheme. For this, we show how to rerandomize ciphertexts, how add and remove layers of encryption, and finally how to homomorphically compute XOR. We remark that it is enough to provide XOR-Homomorphic PKCR encryption scheme to achieve an OR-Homomorphic PKCR encryption scheme, as was shown in [ALM17a]

Rerandomization: We note that a ciphertext can be rerandomized, which is done by homomorphically adding an encryption of 0. The algorithm Rand takes as input a cipertext and the corresponding public key, as well as a (random) vector $\mathbf{x} \in \{0,1\}^m$.

Algorithm $\mathsf{Rand}(c = (c_1, c_2), \mathtt{pk}, \mathbf{x})$

return $(c_1 + \mathbf{x}^\intercal A, c_2 + \mathbf{x}^\intercal \mathtt{pk})$.

Adding and Deleting Layers of Encryption: Given an encryption of a bit b under the public key $\mathtt{pk} = A \cdot \mathtt{sk} + \mathbf{e}$, and a secret key $\mathtt{sk}'$ with corresponding public key $\mathtt{pk}' = A \cdot \mathtt{sk}' + \mathbf{e}'$, one can add a layer of encryption, i.e. obtain a ciphertext under the public key $\mathtt{pk} \cdot \mathtt{pk}' \coloneqq A \cdot (\mathtt{sk} + \mathtt{sk}') + \mathbf{e} + \mathbf{e}'$. Also, one can delete a layer of encryption.

Algorithm $\mathsf{AddLayer}(c = (c_1, c_2), \mathtt{sk})$

return $(c_1, c_1 \cdot \mathtt{sk} + c_2)$

Algorithm $\mathsf{DelLayer}(c = (c_1, c_2), \mathtt{sk})$

return $(c_1, c_2 - c_1 \cdot \mathtt{sk})$

Error Analysis Every time we add a layer, the error increases. Hence, we need to ensure that the error does not increase too much. After l steps, the error in the public key is $\mathtt{pk}_{0\ldots l} = \sum_{i=0}^{l} \mathbf{e}_i$, where $\mathbf{e}_i$ is the error added in each step.
The error in the ciphertext is $c_{0\ldots l} = \sum_{i=0}^{l} \mathbf{x}_i \sum_{j=0}^{i} \mathbf{e}_j$, where the $\mathbf{x}_i$ is the chosen randomness in each step. Since $\mathbf{x}_i \in \{0,1\}^m$, the error in the ciphertext can be bounded by $m \cdot \max_i\{|\mathbf{e}_i|_\infty\} \cdot l^2$, which is quadratic in the number of steps.

Homomorphic XOR: A PKCR encryption scheme requires a slightly stronger version of homomorphism. In particular, homomorphic operation includes the rerandomization of the ciphertexts. Hence, the algorithm hXor also calls Rand. The inputs to hXor are two ciphertexts encrypted under the same public key and the corresponding public key.

Algorithm $\mathsf{hXor}(c = (c_1, c_2), c' = (c_1', c_2'), \mathtt{pk})$

Set $c'' = (c_1 + c_1', c_2 + c_2')$.
Choose $\mathbf{x} \in \{0,1\}^m$ uniformly at random.
return $\mathsf{Rand}(c'', \mathtt{pk}, \mathbf{x})$

References

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Heidelberg, April 2012.

[ALM17a] Adi Akavia, Rio LaVigne, and Tal Moran. Topology-hiding computation on all graphs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 447–467. Springer, Heidelberg, August 2017.

[ALM17b] Adi Akavia, Rio LaVigne, and Tal Moran. Topology-hiding computation on all graphs. Cryptology ePrint Archive, Report 2017/296, 2017. http://eprint.iacr.org/2017/296.

[AM17] Adi Akavia and Tal Moran. Topology-hiding computation beyond logarithmic diameter. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 609–637. Springer, Heidelberg, May 2017.

[BBMM18] Marshall Ball, Elette Boyle, Tal Malkin, and Tal Moran. Exploring the boundaries of topology-hiding computation. In *Eurocrypt'18*, 2018.

[Bd90] Jurjen N. Bos and Bert den Boer. Detection of disrupters in the DC protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 320–327. Springer, Heidelberg, April 1990.

[Cha81] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[Cha88] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, March 2015. doi:10.1007/978-3-662-46497-7_19.

[CNE+14] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J Bernstein, Jake Maskiewicz, Hovav Shacham, Matthew Fredrikson, et al. On the practical exploitability of dual ec in tls implementations. In *USENIX security symposium*, pages 319–335, 2014.

[DDWY90] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. In *31st FOCS*, pages 36–45. IEEE Computer Society Press, October 1990.

[GJ04] Philippe Golle and Ari Juels. Dining cryptographers revisited. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 456–473. Springer, Heidelberg, May 2004.

[HDWH12] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *USENIX Security Symposium*, volume 8, page 1, 2012.

[HJ07] Markus Hinkelmann and Andreas Jakoby. Communications in unknown networks: Preserving the secret of topology. *Theoretical Computer Science*, 384(2-3):184–200, 2007.

[HMTZ16] Martin Hirt, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Network-hiding communication and applications to multi-party protocols. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 335–365. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53008-5_12.

[LLM+18] Rio Lavigne, Chen-Da Liu-Zhang, Ueli Maurer, Tal Moran, Marta Mularczyk, and Daniel Tschudi. Topology-hiding computation beyond semi-honest adversaries. Cryptology ePrint Archive, Report 2018/255, 2018. https://eprint.iacr.org/2018/255.

[MOR15] Tal Moran, Ilan Orlov, and Silas Richelson. Topology-hiding computation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 159–181. Springer, Heidelberg, March 2015. doi:10.1007/978-3-662-46494-6_8.

[RC88] MK Reiter and RA Crowds. Anonymity for web transaction. *ACM Transactions on Information and System Security*, pages 66–92, 1988.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[SGR97] Paul F Syverson, David M Goldschlag, and Michael G Reed. Anonymous connections and onion routing. In *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*, pages 44–54. IEEE, 1997.

Appendix

A Topology-Hiding Broadcast

This section contains supplementary material for Section 3.

A.1 Protocol Leaking One Bit

In this section we prove Theorem 2 from Section 3.1.

Theorem 2. *For κ security parameter and $\mathsf{T} = 8n^3(\log(n)+\kappa)$ protocol $\mathsf{BC\text{-}OB}(\mathsf{T}, (d_i, b_i)_{P_i \in \mathcal{P}})$ topology-hidingly realizes $\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{OB}} || \mathcal{F}_{\text{BC}}$ (with abort) in the $\mathcal{F}_{\text{NET}}$ hybrid-world, where the leakage function $\mathcal{L}_{OB}$ is the one defined as above. If no crashes occur, then there is no abort and there is no leakage.*

Proof. **Completeness.** We first show that the protocol is complete. To this end, we need to ensure that the probability that all parties get the correct output is overwhelming in κ. That is, the probability that all non-dummy random walks (of length $\mathtt{T} = 8n^3(\log(n) + \kappa)$) reach all nodes is overwhelming.

By Lemma 1, a walk of length $8n^3\tau$ does not reach all nodes with probability at most $\frac{1}{2^\tau}$. Then, using the union bound, we obtain that the probability that there is a party whose walk does not reach all nodes is at most $\frac{n}{2^\tau}$. Hence, all n walks (one for each party) reach all nodes with probability at least $1 - \frac{n}{2^\tau}$. If we want this value to be overwhelming, e.g. $1 - \frac{1}{2^\kappa}$, we can set $\tau \coloneqq \kappa + \log(n)$.

Soundness. We now need to show that no environment can distinguish between the real world and the simulated world, when given access to the adversarially-corrupted parties. We first describe on a high level the simulator $\mathcal{S}_{OB}$ and argue that it simulates the real execution.

In essence, the task of $\mathcal{S}_{OB}$ is to simulate the messages sent by honest parties to passively corrupted parties. Consider a corrupted party P_c and its honest neighbor P_h. The messages sent from P_h to P_c during the Aggregate Stage are ciphertexts, to which P_h added a layer, and corresponding public keys. Since P_h is honest, the adversary does not know the secret keys corresponding to the sent public keys. Hence, $\mathcal{S}_{OB}$ can simply replace them with encryptions of a pair $(1, 1)$ under a freshly generated public key. The group structure of keys in PKCR guarantees that a fresh key has the same distribution as the composed key (after executing $\mathsf{AddLayer}$). Semantic security implies that the encrypted message can be replaced by $(1, 1)$.

Consider now the Decrypt Stage at round r. Let $\mathtt{pk}_{c\to h}^{(r)}$ be the public key sent by P_c to P_h in the Aggregate Stage (note that this is not the key discussed above; there we argued about keys sent in the opposite direction). $\mathcal{S}_{OB}$ will send to P_c a fresh encryption under $\mathtt{pk}_{c\to h}^{(r)}$. We now specify what it encrypts.

Note that the only interesting case is when the party P_o receiving output is corrupted and when we are in the round r in which the (only one) random walk carrying the output enters an area of corrupted parties, containing P_o (that is, when the walk with output contains from P_h all the way to P_o only corrupted parties). In this one message in round r the adversary learns the output of P_o. All other messages are simply encryptions of $(1, 1)$.

For this one meaningful message, we consider three cases. If any party crashed in a phase preceding the current one, $\mathcal{S}_{OB}$ sends an encryption of $(1, 1)$ (as in the real world the walk is made dummy by an unhappy party). If no crashes occurred up to this point (round r in given phase), $\mathcal{S}_{OB}$ encrypts the output received from $\mathcal{F}_{\text{BC}}$. If a crash happened in the given phase, $\mathcal{S}_{OB}$ queries the leakage oracle $\mathcal{L}_{OB}$, which essentially executes the protocol and tells whether the output or $(1, 1)$ should be sent.

Simulator. Below, we present the pseudocode of the simulator. The essential part of it is the algorithm $\mathsf{PhaseSimulation}$, which is also illustrated in Figure 1.

Simulator $\mathcal{S}_{OB}$

1. $\mathcal{S}_{OB}$ corrupts passively $\mathcal{Z}^p$.
2. $\mathcal{S}_{OB}$ sends inputs for all parties in $\mathcal{Z}^p$ to $\mathcal{F}_{\text{BC}}$ and receives the output bit b^{out}.
3. For each $P_i \in \mathcal{Z}^p$, $\mathcal{S}_{OB}$ receives $\mathbf{N}_G(P_i)$ from $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$.
4. Throughout the simulation, if $\mathcal{A}$ crashes a party P_f, so does $\mathcal{S}_{OB}$.
5. Now $\mathcal{S}_{OB}$ has to simulate the view of all parties in $\mathcal{Z}^p$.
   In every phase in which P_o should get the output, first of all the Initialization Stage is executed among the parties in $\mathcal{Z}^p$ and the $\mathtt{T}$ key pairs are generated for every $P_i \in \mathcal{Z}^p$. Moreover, for every $P_i \in \mathcal{Z}^p$ the permutations $\pi_i^{(r)}$ are generated, defining those parts of all random walks, which pass through parties in $\mathcal{Z}^p$.
   The messages sent by parties in $\mathcal{Z}^p$ are generated by executing the protocol RandomWalkPhase. The messages sent by correct parties $P_i \notin \mathcal{Z}^p$ are generated by executing PhaseSimulation(P_o, P_i), described below.
6. $\mathcal{S}_{OB}$ sends to $\mathcal{F}_{\text{BC}}$ the abort vector (in particular, the vector contains all parties P_o who should receive their outputs in phases following the first crash and, depending on the output of $\mathcal{L}_{OB}$, the party who should receive its output in the phase with first crash).

Algorithm PhaseSimulation(P_o, P_i)

If $P_o \in \mathcal{Z}^p$, let w denote the random walk generated in the Initialization Stage (at the beginning of the simulation of this phase), which starts at P_o and carries the output bit. Let ℓ denote the number of parties in $\mathcal{Z}^p$ on w before the first correct party. If $P_o \notin \mathcal{Z}^p$, w and ℓ are not defined.

For every $P_j \in \mathcal{Z}^p \cap \mathbf{N}_G(P_i)$, let $\mathtt{pk}_{j\to i}^{(r)}$ denote the public key generated in the Initialization Stage by P_j for P_i and for round r.

Initialization Stage

1: For every neighbor $P_j \in \mathcal{Z}^p$ of the correct P_i, $\mathcal{S}_{OB}$ generates $\mathtt{T}$ key pairs $(\mathtt{pk}_{i\to j}^{(1)}, \mathtt{sk}_{i\to j}^{(1)}), \ldots, (\mathtt{pk}_{i\to j}^{(\mathtt{T})}, \mathtt{sk}_{i\to j}^{(\mathtt{T})})$.

Aggregate Stage

1: In round r, for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{OB}$ sends $([1,1]_{\mathtt{pk}_{i\to j}^{(r)}}, \mathtt{pk}_{i\to j}^{(r)})$ to P_j.

Decrypt Stage

1: **if** $\mathcal{A}$ crashed any party in any phase before the current one or $P_o \notin \mathcal{Z}^p$ **then**
2: In every round r and for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{OB}$ sends $[1,1]_{\mathtt{pk}_{j\to i}^{(r)}}$ to P_j.
3: **else**
4: In every round r and for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{OB}$ sends $[1,1]_{\mathtt{pk}_{j\to i}^{(r)}}$ to P_j unless the following three conditions hold: (a) P_i is the first party not in $\mathcal{Z}^p$ on w, (b) P_j is the last party in $\mathcal{Z}^p$ on w, and (c) $r = 2\mathtt{T} - \ell$.
5: If the three conditions hold (in particular $r = 2\mathtt{T} - \ell$), $\mathcal{S}_{OB}$ does the following. If $\mathcal{A}$ did not crash any party in a previous round, $\mathcal{S}_{OB}$ sends $[b^{out}, 0]_{\mathtt{pk}_{j\to i}^{(r)}}$ to party P_j.
6: Otherwise, let F denote the set of pairs $(P_f, s - \ell + 1)$ such that $\mathcal{A}$ crashed P_f in round s. $\mathcal{S}_{OB}$ queries $\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{OB}}$ for the leakage on input $(F, P_i, \mathtt{T} - \ell)$. If the returned value is 1, it sends $[1,1]_{\mathtt{pk}_{j\to i}^{(r)}}$ to P_j. Otherwise it sends $[b^{out}, 0]_{\mathtt{pk}_{j\to i}^{(r)}}$ to party P_j.
7: **end if**

We prove that no environment can tell whether it is interacting with $\mathcal{F}_{\text{NET}}$ and the adversary in the real world or with $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ and the simulator in the ideal world.

Hybrids and security proof.

Hybrid 1. $\mathcal{S}_1$ simulates the real world exactly. This means, $\mathcal{S}$ has information on the entire topology of the graph, each party's input, and can simulate identically the real world.

Hybrid 2. $\mathcal{S}_2$ replaces the real keys with the simulated public keys, but still knows everything about the graph as in the first hybrid.
More formally, in each random walk phase and for each party $P_i \in \mathcal{P} \setminus \mathcal{Z}^p$ where $\mathbf{N}_G(P_i) \cap \mathcal{Z}^p \neq \varnothing$, $\mathcal{S}_2$ generates T key pairs $(\mathtt{pk}_{i\to j}^{(1)}, \mathtt{sk}_{i\to j}^{(1)}), \ldots, (\mathtt{pk}_{i\to j}^{(\mathtt{T})}, \mathtt{sk}_{i\to j}^{(\mathtt{T})})$ for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$. In each round r of the corresponding Aggregate Stage and for every

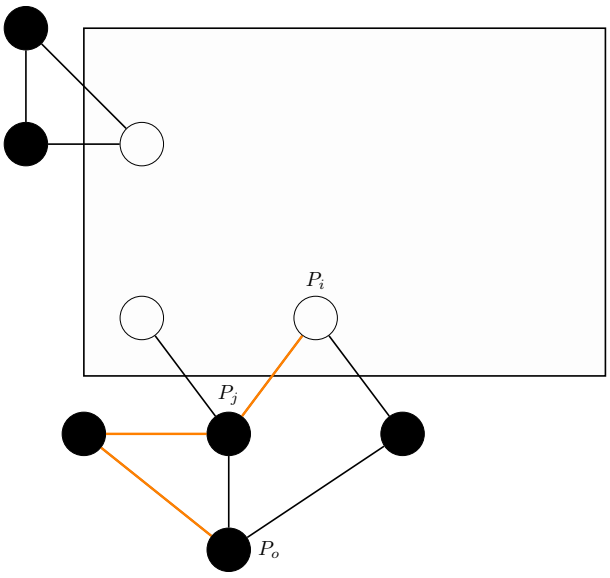

Fig. 1. An example of the algorithm executed by the simulator $\mathcal{S}_{OB}$. The filled circles are the corrupted parties. The red line represents the random walk generated by $\mathcal{S}_{OB}$ in Step 5, in this case of length $\ell = 3$. $\mathcal{S}_{OB}$ simulates the Decrypt Stage by sending fresh encryptions of $(1, 1)$ at every round from every honest party to each of its corrupted neighbors, except in round $2\mathtt{T} - 3$ from P_i to P_j. If no crash occurred up to that point, $\mathcal{S}_{OB}$ sends encryption of $(b^{out}, 0)$. Otherwise, it queries the leakage oracle about the walk of length $\mathtt{T} - 3$, starting at P_i.

neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_2$ does the following. P_i receives ciphertext $[b, u]_{\mathtt{pk}^{(r)}_{*\to i}}$ and the public key $\mathtt{pk}^{(r)}_{*\to i}$ destined for P_j. Instead of adding a layer and homomorphically OR'ing the bit b_i, $\mathcal{S}_2$ computes $(b', u') = (b \vee b_i \vee u_i, u \vee u_i)$, and sends $[b', u']_{\mathtt{pk}^{(r)}_{i\to j}}$ to P_j. In other words, it sends the same message as $\mathcal{S}_1$ but encrypted with a fresh public key. In the corresponding Decrypt Stage, P_i will get back a ciphertext from P_j encrypted under this exact fresh public key.

Hybrid 3. $\mathcal{S}_3$ now simulates the ideal functionality during the Aggregate Stage. It does so by sending encryptions of $(1, 1)$ instead of the actual messages and unhappy bits. More formally, in each round r of the Aggregate Stage and for all parties $P_i \in \mathcal{P} \setminus \mathcal{Z}^p$ and $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_3$ sends $[1, 1]_{\mathtt{pk}^{(r)}_{i\to j}}$ instead of the ciphertext $[b, u]_{\mathtt{pk}^{(r)}_{i\to j}}$ sent by $\mathcal{S}_2$.

Hybrid 4. $\mathcal{S}_4$ does the same as $\mathcal{S}_{OB}$ during the Decrypt Stage for all steps except for round $2\mathtt{T}-\ell$ of the first random walk phase in which the adversary crashes a party. This corresponds to the original description of the simulator except for the 'Otherwise' condition of Step 6 in the Decrypt Stage.

Hybrid 5. $\mathcal{S}_5$ is the actual simulator $\mathcal{S}_{OB}$.

In order to prove that no environment can distinguish between the real world and the ideal world, we prove that no environment can distinguish between any two consecutive hybrids when given access to the adversarially-corrupted nodes.

Claim 1. *No efficient distinguisher D can distinguish between Hybrid 1 and Hybrid 2.*

Proof: The two hybrids only differ in the computation of the public keys that are used to encrypt messages in the Aggregate Stage from any honest party $P_i \in \mathcal{P} \setminus \mathcal{Z}^p$ to any dishonest neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$.

In Hybrid 1, party P_i sends to P_j an encryption under a fresh public key in the first round. In the following rounds, the encryption is sent either under a product key $\overline{\mathtt{pk}}^{(r)}_{i\to j} = \overline{\mathtt{pk}}^{(r-1)}_{k\to i} \circledast \mathtt{pk}^{(r)}_{i\to j}$ or under a fresh public key (if P_i is unhappy). Note that $\overline{\mathtt{pk}}^{(r-1)}_{k\to i}$ is the key P_i received from a neighbor P_k in the previous round.

In Hybrid 2, party P_i sends to P_j an encryption under a fresh public key $\mathtt{pk}_{i\to j}^{(r)}$ in every round.

The distribution of the product key used in Hybrid 1 is the same as the distribution of a freshly generated public-key. This is due to the (fresh) $\mathtt{pk}_{i\to j}^{(r)}$ key which randomizes the product key. Therefore, no distinguisher can distinguish between Hybrid 1 and Hybrid 2. ■

Claim 2. *No efficient distinguisher D can distinguish between Hybrid 2 and Hybrid 3.*

Proof: The two hybrids differ only in the content of the encrypted messages that are sent in the Aggregate Stage from any honest party $P_i \in \mathcal{P}\setminus\mathcal{Z}^p$ to any dishonest neighbor $P_j \in \mathbf{N}_G(P_i)\cap\mathcal{Z}^p$.

In Hybrid 2, party P_i sends to P_j in the first round an encryption of $(b_i \vee u_i, u_i)$. In the following rounds, P_i sends to P_j either an encryption of $(b \vee b_i \vee u_i, u \vee u_i)$, if message (b, u) is received from neighbor $\pi_i^{-1}(j)$, or an encryption of $(1, 1)$ if no message is received.

In Hybrid 3, all encryptions that are sent from party P_i to party P_j are replaced by encryptions of $(1, 1)$.

Since the simulator chooses a key independent of any key chosen by parties in $\mathcal{Z}^p$ in each round, the key is unknown to the adversary. Hence, the semantic security of the encryption scheme guarantees that the distinguisher cannot distinguish between both encryptions. ■

Claim 3. *No efficient distinguisher D can distinguish between Hybrid 3 and Hybrid 4.*

Proof: The only difference between the two hybrids is in the Decrypt Stage. We differentiate two cases:

- A phase where the adversary did not crash any party in this or any previous phase. In this case, the simulator $\mathcal{S}_3$ sends an encryption of (b_W, u_W), where $b_W = \bigvee_{P_j\in W} b_j$ is the OR of all input bits in the walk and $u_W = 0$, since no crash occurred. $\mathcal{S}_4$ sends an encryption of $(b^{out}, 0)$, where $b^{out} = \bigvee_{P_i\in\mathcal{P}} b_i$. Since the graph is connected, $b^{out} = b_W$ with overwhelming probability, as proven in Corollary 1. Also, the encryption in Hybrid 4 is done with a fresh public key which is indistinguishable with the encryption done in Hybrid 3 by OR'ing many times in the graph, as shown in Claim 2.1 in [ALM17a].
- A phase where the adversary crashed a party in a previous phase or any round different than $2\mathtt{T} - \ell$ of the first phase where the adversary crashes a party. In Hybrid 4 the parties send an encryption of $(1, 1)$. This is also the case in Hybrid 3, because even if a crashed party disconnected the graph, each connected component contains a neighbor of a crashed party. Moreover, in Hybrid 4, the messages are encrypted with a fresh public key, and in Hybrid 3, the encryptions are obtained by the homomorphic OR operation. Both encryptions are indistinguishable, as shown in in Claim 2.1 in [ALM17a].

■

Claim 4. *No efficient distinguisher D can distinguish between Hybrid 4 and Hybrid 5.*

Proof: The only difference between the two hybrids is in the Decrypt Stage, at round $2\mathtt{T} - \ell$ of the first phase where the adversary crashes.

Let F be the set of pairs (P_f, r) such that $\mathcal{A}$ crashed P_f at round r of the phase. In Hybrid 4, a walk W of length $\mathtt{T}$ is generated from party P_o. Let W_1 be the region of W from P_o to the first not passively corrupted party and let W_2 be the rest of the walk. Then, the adversary's view at this step is the encryption of $(1, 1)$ if one of the crashed parties breaks W_2, and otherwise an encryption of $(b_W, 0)$. In both cases, the message is encrypted under a public key for which the adversary knows the secret key.

In Hybrid 5, a walk W_1' is generated from P_o of length $\ell \leq \mathtt{T}$ ending at the first not passively corrupted party P_i. Then, the simulator queries the leakage function on input $(F, P_i, \mathtt{T} - \ell)$, which generates a walk W_2' of length $\mathtt{T} - \ell$ from P_i, and checks whether W_2' is broken by any

party in F. If W_2' is broken, P_i sends an encryption of $(1,1)$, and otherwise an encryption of $(b_W, 0)$. Since the walk W' defined as W_1' followed by W_2' follows the same distribution as W, $b_W = b^{out}$ with overwhelming probability, and the encryption with a fresh public key which is indistinguishable with the encryption done by OR'ing many times in the graph, then it is impossible to distinguish between Hybrid 4 and Hybrid 5.

■

This concludes the proof of soundness. □

A.2 Protocol Leaking a Fraction of a Bit

In this section, we give a formal description of $\mathsf{ProbabilisticRandomWalkPhase}_p$ which is the random-walk phase protocol for the broadcast protocol $\mathsf{BC\text{-}FB}_p$, from Section 3.2. Note that this protocol should be repeated ρ times in the actual protocol. The boxes indicate the parts where it differs from the random-walk phase protocol RandomWalkPhase for the broadcast protocol leaking one bit (cf. Section 3.1).

Protocol $\mathsf{ProbabilisticRandomWalkPhase}_p(\mathtt{T}, P_o, (d_i, b_i, u_i)_{P_i \in \mathcal{P}})$

Initialization Stage:

1: Each party P_i generates $\mathtt{T} \cdot d_i$ keypairs $(\mathtt{pk}_{i\to j}^{(r)}, \mathtt{sk}_{i\to j}^{(r)}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ where $r \in \{1, \ldots, \mathtt{T}\}$ and $j \in \{1, \ldots, d_i\}$.

2: Each party P_i generates $\mathtt{T} - 1$ random permutations on d_i elements $\left\{\pi_i^{(2)}, \ldots, \pi_i^{(\mathtt{T})}\right\}$

3: For each party P_i, if any of P_i's neighbors crashed in any phase before the current one, then P_i becomes unhappy, i.e., sets $u_i = 1$.

Aggregate Stage: Each party P_i does the following:

1: **if** P_i is the recipient P_o **then**

2: Party P_i sends to the first neighbor the public key $\mathtt{pk}_{i\to 1}^{(1)}$ and the ciphertext $\boxed{[b_i \vee u_i, 1, \ldots, 1, u_i]_{\mathtt{pk}_{i\to 1}^{(1)}}}$ ($\lfloor 1/p \rfloor - 1$ ciphertexts contain 1), and to any other neighbor P_j it sends $\boxed{[1, \ldots, 1, 1]_{\mathtt{pk}_{i\to j}^{(1)}}}$ and the public key $\mathtt{pk}_{i\to j}^{(1)}$.

3: **else**

4: Party P_i sends to each neighbor P_j ciphertext $\boxed{[1, \ldots, 1, 1]_{\mathtt{pk}_{i\to j}^{(1)}}}$ and the public key $\mathtt{pk}_{i\to j}^{(1)}$.

5: **end if**

6: // Add layer while ORing own input bit

7: **for** any round r from 2 to $\mathtt{T}$ **do**

8: For each neighbor P_j of P_i, do the following (let $k = \pi_i^{(r)}(j)$):

9: **if** P_i did not receive a message from P_j **then**

10: Party P_i sends $\boxed{[1, \ldots, 1, 1]_{\mathtt{pk}_{i\to k}^{(r)}}}$ and $\mathtt{pk}_{i\to k}^{(r)}$ to neighbor P_k.

11: **else**

12: Let $\mathtt{c}_{j\to i}^{(r-1)}$ and $\overline{\mathtt{pk}}_{j\to i}^{(r-1)}$ be the ciphertext and the public key P_i received from P_j. Party P_i computes $\overline{\mathtt{pk}}_{i\to k}^{(r)} = \overline{\mathtt{pk}}_{j\to i}^{(r-1)} \circledast \mathtt{pk}_{i\to k}^{(r)}$ and $\hat{\mathtt{c}}_{i\to k}^{(r)} \leftarrow \mathsf{AddLayer}\left(\mathtt{c}_{j\to i}^{(r-1)}, \mathtt{pk}_{i\to k}^{(r)}\right)$.

13: Party P_i computes $\boxed{[b_i \vee u_i, \ldots, b_i \vee u_i, u_i]_{\overline{\mathtt{pk}}_{i\to k}^{(r)}}}$ and $\boxed{\mathtt{c}_{i\to k}^{(r)} = \mathsf{HomOR}\left([b_i \vee u_i, \ldots, b_i \vee u_i, u_i]_{\overline{\mathtt{pk}}_{i\to k}^{(r)}}, \hat{\mathtt{c}}_{i\to k}^{(r)}\right)}$.

14: Party P_i sends ciphertext $\mathtt{c}_{i\to k}^{(r)}$ and public key $\overline{\mathtt{pk}}_{i\to k}^{(r)}$ to neighbor P_k.

15: **end if**

16: **end for**

Decrypt Stage: Each party P_i does the following:

1: For each neighbor P_j of P_i:

2: **if** P_i did not receive a message from P_j at round T of the Aggregate Stage **then**

3: Party P_i sends ciphertext $\mathsf{e}_{i\to j}^{(\mathtt{T})} = [1,1]_{\overline{\mathsf{pk}}_{j\to i}^{(\mathtt{T})}}$ to P_j.
4: **else**
5: Party P_i chooses uniformly at random one of the first $\lfloor 1/p \rfloor$ ciphertexts in $\mathsf{c}_{j\to i}^{(\mathtt{T})}$. Let $\bar{\mathsf{c}}_{j\to i}^{(\mathtt{T})}$ denote the tuple containing the chosen ciphertext and the last element of $\mathsf{c}_{j\to i}^{(\mathtt{T})}$ (the encryption of the unhappy bit). Party P_i computes and sends $\mathsf{e}_{i\to j}^{(\mathtt{T})} = \mathsf{HomOR}\left([b_i \vee u_i, u_i]_{\overline{\mathsf{pk}}_{j\to i}^{(\mathtt{T})}}, \bar{\mathsf{c}}_{j\to i}^{(\mathtt{T})}\right)$ to P_j.
6: **end if**
7: **for** any round r from $\mathtt{T}$ to 2 **do**
8: For each neighbor P_k of P_i:
9: **if** P_i did not receive a message from P_k **then**
10: Party P_i sends $\mathsf{e}_{i\to j}^{(r-1)} = [1,1]_{\overline{\mathsf{pk}}_{j\to i}^{(r-1)}}$ to neighbor P_j, where $k = \pi_i^{(r)}(j)$.
11: **else**
12: Denote by $\mathsf{e}_{k\to i}^{(r)}$ the ciphertext P_i received from P_k, where $k = \pi_i^{(r)}(j)$. Party P_i sends $\mathsf{e}_{i\to j}^{(r-1)} = \mathsf{DelLayer}\left(\mathsf{e}_{k\to i}^{(r)}, \mathsf{sk}_{i\to k}^{(r)}\right)$ to neighbor P_j.
13: **end if**
14: **end for**
15: If P_i is the recipient P_o, then it computes $(b,u) = \mathsf{Decrypt}(\mathsf{e}_{1\to i}^{(1)}, \mathsf{sk}_{i\to 1}^{(1)})$ and **outputs** (b,u,u_i). Otherwise, it **outputs** $(1,0,u_i)$.

Security Proof of the Protocol Leaking a Fraction of a Bit.

In this section we prove Theorem 3 from Section 3.2.

Theorem 3. *Let κ be the security parameter. For $\tau = \log(n) + \kappa$, $\mathtt{T} = 8n^3\tau$ and $\rho = \tau/(p' - 2^{-\tau})$, where $p' = 1/\lfloor 1/p \rfloor$, the protocol $\mathsf{BC\text{-}FB}_p(\mathtt{T}, \rho, (d_i, b_i)_{P_i\in\mathcal{P}})$ topology-hidingly realizes the functionalities $\mathcal{F}_{\mathrm{INFO}}^{\mathcal{L}_{FB_p}} || \mathcal{F}_{\mathrm{BC}}$ (with abort) in the $\mathcal{F}_{\mathrm{NET}}$ hybrid-world, where the leakage function $\mathcal{L}_{FB_p}$ is the one defined as above. If no crashes occur, then there is no abort and there is no leakage.*

Proof. **Completeness.** We first show that the protocol is complete. That is, that if the adversary does not crash any party, then every party gets the correct output (the OR of all input bits) with overwhelming probability. More specifically, we show that if no crashes occur, then after ρ repetitions of a phase, the party P_o outputs the correct value with probability at least $1 - 2^{-(\kappa+\log(n))}$. The overall completeness follows from the union bound: the probability that all n parties output the correct value is at least $1 - 2^{-\kappa}$.

Notice that if the output of any of the ρ repetitions intended for P_o is correct, then the overall output of P_o is correct. A given repetition can only give an incorrect output when either the random walk does not reach all parties, which happens with probability at most $2^{-\tau}$, or when the repetition fails, which happens with probability $1 - p'$. Hence, the probability that a repetition gives the incorrect result is at most $1-p'+2^{-\tau}$. The probability that all repetitions are incorrect is then at most $(1-p'+2^{-\tau})^{\rho} \leq 2^{-(\kappa+\log(n))}$ (the inequality holds for $0 \leq p'-2^{-\tau} \leq 1$).

Soundness. We show that no environment can distinguish between the real world and the simulated world, when given access to the adversarially-corrupted nodes. The simulator $\mathcal{S}_{FB}$ for $\mathsf{BC\text{-}FB}_p$ is a modification of $\mathcal{S}_{OB}$. Here we only sketch the changes and argue why $\mathcal{S}_{FB}$ simulates the real world.

In each of the ρ repetitions of a phase, $\mathcal{S}_{FB}$ executes a protocol very similar to the one for $\mathcal{S}_{OB}$. In the Aggregate Stage, $\mathcal{S}_{FB}$ proceeds almost identically to $\mathcal{S}_{OB}$ (except that it sends encryptions of vectors $(1,\ldots,1)$ instead of only two values). In the Decrypt Stage the only difference between $\mathcal{S}_{FB}$ and $\mathcal{S}_{OB}$ is in computing the output for the party P_o (as already discussed in the proof of Theorem 2, $\mathcal{S}_{FB}$ does this only when P_o is corrupted and the walk carrying the output enters an area of corrupted parties). In the case when there were no crashes before or during given repetition of a phase, $\mathcal{S}_{OB}$ would simply send the encrypted output. On the other hand, $\mathcal{S}_{FB}$ samples a value from the Bernoulli distribution with parameter p and

sends the encrypted output only with probability p, while with probability $1-p$ it sends the encryption of $(1,0)$. Otherwise, the simulation is the same as for $\mathcal{S}_{OB}$.

It can be easily seen that $\mathcal{S}_{FB}$ simulates the real world in the Aggregate Stage and in the Decrypt Stage in every message other than the one encrypting the output. But even this message comes from the same distribution as the corresponding message sent in the real world. This is because in the real world, if the walk was not broken by a crash, this message contains the output with probability p. The simulator encrypts the output also with probability p in the two possible cases: when there was no crash ($\mathcal{S}_{FB}$ samples from the Bernoulli distribution) and when there was a crash but the walk was not broken ($\mathcal{L}_{FB}$ is defined in this way).

Simulator. The simulator $\mathcal{S}_{FB}$ proceeds almost identically to the simulator $\mathcal{S}_{OB}$ given in the proof of Theorem 2 (cf. Section A.1). We only change the algorithm PhaseSimulation to ProbabilisticPhaseSimulation and execute it ρ times instead of only once.

Algorithm ProbabilisticPhaseSimulation(P_o, P_i)

If $P_o \in \mathcal{Z}^p$, let w denote the random walk generated in the Initialization Stage (at the beginning of the simulation of this phase), which starts at P_o and carries the output bit. Let ℓ denote the number of parties in $\mathcal{Z}^p$ on w before the first correct party. If $P_o \notin \mathcal{Z}^p$, w and ℓ are not defined.

For every $P_j \in \mathcal{Z}^p \cap \mathbf{N}_G(P_i)$, let $\mathtt{pk}^{(r)}_{j\to i}$ denote the public key generated in the Initialization Stage by P_j for P_i and for round r.

Initialization Stage

1: For every neighbor $P_j \in \mathcal{Z}^p$ of the correct P_i, $\mathcal{S}_{FB}$ generates $\mathtt{T}$ key pairs $(\mathtt{pk}^{(1)}_{i\to j},\mathtt{sk}^{(1)}_{i\to j}),\ldots,(\mathtt{pk}^{(\mathtt{T})}_{i\to j},\mathtt{sk}^{(\mathtt{T})}_{i\to j})$.

Aggregate Stage

1: In round r, for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{FB}$ sends the tuple $([1,\ldots,1]_{\mathtt{pk}^{(r)}_{i\to j}}, \mathtt{pk}^{(r)}_{i\to j})$ (with $\lfloor 1/p \rfloor + 1$ ones) to P_j.

Decrypt Stage

1: **if** $P_o \notin \mathcal{Z}^p$ or $\mathcal{A}$ crashed any party in any phase before the current one
2: or in any repetition of the current phase **then**
3: In every round r and for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{FB}$ sends $[1,1]_{\mathtt{pk}^{(r)}_{j\to i}}$ to P_j.
4: **else**
5: In every round r and for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{FB}$ sends $[1,1]_{\mathtt{pk}^{(r)}_{j\to i}}$ to P_j unless the following three conditions hold: (a) P_i is the first party not in $\mathcal{Z}^p$ on w, (b) P_j is the last party in $\mathcal{Z}^p$ on w, and (c) $r = 2\mathtt{T} - \ell$.
6: If the three conditions hold (in particular $r = 2\mathtt{T} - \ell$), $\mathcal{S}_{FB}$ does the following. If $\mathcal{A}$ did not crash any party in a previous round,
7: $\mathcal{S}_{FB}$ samples a value x from the Bernoulli distribution with parameter p'. If $x = 1$ (with probability p'), $\mathcal{S}_{FB}$ sends to P_j the ciphertext $[b_{out}, 0]_{pk^{(r)}_{j\to i}}$ and otherwise it sends $[1,0]_{pk^{(r)}_{j\to i}}$.
8: Otherwise, let F denote the set of pairs $(P_f, s-\ell+1)$ such that $\mathcal{A}$ crashed P_f in round s. $\mathcal{S}_{FB}$ queries $\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{FB_p}}$ for the leakage on input $(F, P_i, \mathtt{T}-\ell)$. If the returned value is 1, it sends $[1,1]_{\mathtt{pk}^{(r)}_{j\to i}}$ to P_j. Otherwise it sends $[b^{out}, 0]_{\mathtt{pk}^{(r)}_{j\to i}}$ to party P_j.
9: **end if**

Hybrids and security proof. We consider similar steps as the hybrids from Paragraph A.1.

Hybrid 1. $\mathcal{S}_1$ simulates the real world exactly. This means, $\mathcal{S}_1$ has information on the entire topology of the graph, each party's input, and can simulate identically the real world.

Hybrid 2. $\mathcal{S}_2$ replaces the real keys with the simulated public keys, but still knows everything about the graph as in the first hybrid.

More formally, in each subphase of each random walk phase and for each party $P_i \in \mathcal{P} \setminus \mathcal{Z}^p$ where $\mathbf{N}_G(P_i) \cap \mathcal{Z}^p \neq \varnothing$, $\mathcal{S}_2$ generates T key pairs $(\mathtt{pk}_{i\to j}^{(1)}, \mathtt{sk}_{i\to j}^{(1)}), \ldots, (\mathtt{pk}_{i\to j}^{(\mathtt{T})}, \mathtt{sk}_{i\to j}^{(\mathtt{T})})$ for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$. Let $\alpha \coloneqq \lfloor \frac{1}{p} \rfloor$. In each round r of the corresponding Aggregate Stage and for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_2$ does the following: P_i receives ciphertext $[b_1, \ldots, b_\alpha, u]_{\mathtt{pk}_{*\to i}^{(r)}}$ and the public key $\mathtt{pk}_{*\to i}^{(r)}$ destined for P_j. Instead of adding a layer and homomorphically OR'ing the bit b_i, $\mathcal{S}_2$ computes $(b'_1, \ldots, b'_\alpha, u') = (b_1 \vee b_i \vee u_i, \cdots, b_\alpha \vee b_i \vee u_i, u \vee u_i)$, and sends $[b'_{\sigma(1)}, \cdots, b'_{\sigma(\alpha)}, u']_{\mathtt{pk}_{i\to j}^{(r)}}$ to P_j, where σ is a random permutation on α elements. In other words, it sends the same message as $\mathcal{S}_1$ but encrypted with a fresh public key. In the corresponding Decrypt Stage, P_i will get back a ciphertext from P_j encrypted under this exact fresh public key.

Hybrid 3. $\mathcal{S}_3$ now simulates the ideal functionality during the Aggregate Stage. It does so by sending encryptions of $(1, \ldots, 1)$ instead of the actual messages and unhappy bits. More formally, let $\alpha \coloneqq \lfloor \frac{1}{p} \rfloor$. In each round r of a subphase of a random walk phase and for all parties $P_i \in \mathcal{P} \setminus \mathcal{Z}^p$ and $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_3$ sends $[1, 1, \ldots, 1]_{\mathtt{pk}_{i\to j}^{(r)}}$ instead of the ciphertext $[b_1, \ldots, b_\alpha, u]_{\mathtt{pk}_{i\to j}^{(r)}}$ sent by $\mathcal{S}_2$.

Hybrid 4. $\mathcal{S}_4$ does the same as $\mathcal{S}_{FB}$ during the Decrypt Stage for all phases and subphases except for the first subphase of a random walk phase in which the adversary crashes a party.

Hybrid 5. $\mathcal{S}_5$ is the actual simulator $\mathcal{S}_{FB}$.

The proofs that no efficient distinguisher D can distinguish between Hybrid 1, Hybrid 2 and Hybrid 3 are similar to the Claim 1 and Claim 2. Hence, we prove indistinguishability between Hybrid 3, Hybrid 4 and Hybrid 5.

Claim 5. *No efficient distinguisher D can distinguish between Hybrid 3 and Hybrid 4.*

Proof: The only difference between the two hybrids is in the Decrypt Stage. We differentiate three cases:

- A subphase l of a phase k where the adversary did not crash any party in this phase, any previous subphase, or any previous phase. In this case, $\mathcal{S}_3$ sends with probability p an encryption of (b_W, u_W), where $b_W = \bigvee_{u \in W} b_u$ is the OR of all input bits in the walk and $u_W = 0$ (since no crash occurs), and with probability $1 - p$ an encryption of $(1, 0)$. On the other hand, $\mathcal{S}_4$ samples r from a Bernoulli distribution with parameter p, and if $r = 1$, it sends an encryption of $(b_{out}, 0)$, where $b_{out} = \bigvee_{i \in [n]} b_i$, and if $r = 0$ it sends an encryption of $(1, 0)$. Since the graph is connected, $b_{out} = b_W$ with overwhelming probability, as proven in Corollary 1. Also, the encryption in Hybrid 4 is done with a fresh public key which is indistinguishable with the encryption done in Hybrid 3 by OR'ing many times in the graph, as shown in Claim 2.1. in [ALM17a].
- A subphase l of a phase k where the adversary crashed a party in a previous subphase or a previous phase.
  In Hybrid 3 the parties send encryptions of $(1, 1)$. This is also the case in Hybrid 4, because even if a crashed party disconnected the graph, each connected component contains a neighbor of a crashed party. Moreover, in Hybrid 4, the messages are encrypted with a fresh public key, and in Hybrid 3, the encryptions are obtained by the homomorphic OR operation. Both encryptions are indistinguishable, as shown in Claim 2.1. in [ALM17a].

■

Claim 6. *No efficient distinguisher D can distinguish between Hybrid 4 and Hybrid 5.*

Proof: The only difference between the two hybrids is in the Decrypt Stage of the first subphase of a phase where the adversary crashes.

Let F be the set of pairs (P_f, r) such that $\mathcal{A}$ crashed P_f at round r of the phase. In Hybrid 4, a walk W of length $\mathtt{T}$ is generated from party P_o. Let W_1 be the region of W from P_o to the first not passively corrupted party and let W_2 be the rest of the walk. Then, the adversary's view at this step is the encryption of $(1,1)$ if one of the crashed parties breaks W_2 or if the walk became dummy (which happens with probability $1-p$, since the ciphertexts are permuted randomly and only one ciphertext out of $\frac{1}{p}$ contains b_W). Otherwise, the adversary's view is an encryption of $(b_W, 0)$. In both cases, the message is encrypted under a public key for which the adversary knows the secret key.

In Hybrid 5, a walk W_1' is generated from P_o of length $\ell \leq \mathtt{T}$ ending at the first not passively corrupted party P_i. Then, the simulator queries the leakage function on input $(F, P_i, \mathtt{T} - \ell)$. Then, with probability p it generates a walk W_2' of length $\mathtt{T} - \ell$ from P_i, and checks whether W_2' is broken by any party in F. If W_2' is broken, P_i sends an encryption of $(1,1)$, and otherwise an encryption of $(b_W, 0)$. Since the walk W' defined as W_1' followed by W_2' follows the same distribution as W, $b_W = b^{out}$ with overwhelming probability, and the encryption with a fresh public key which is indistinguishable with the encryption done by OR'ing many times in the graph, then it is impossible to distinguish between Hybrid 4 and Hybrid 5. ■

This concludes the proof of soundness. □

B From Broadcast to Topology-Hiding Computation

This section contains supplementary material for Section 4.

Naive composition of broadcast. We first argue that composing t broadcasts with one bit leakage can in general leak t bits.

Given black-box access to a fail-stop secure topology-hiding broadcast with a leakage function, the naive thing to do to compose broadcasts is run both broadcasts, either in parallel or sequentially. So, consider composing two broadcasts together, first in parallel. Each protocol is running independently, and so if there is an abort, the simulator will need to query the leakage function twice, unless we can make the specific claim that the leakage function will output a correlated bit for independent instances given the same abort (note that our construction does not have this property).

If we run the protocols sequentially, we'll need to make a similar claim. If we are simulating this composition and there is both an abort in the first broadcast and the second, then we definitely need to query the leakage function for the first abort. Then, unless we can make specific claims about how we could start a broadcast protocol *after* there has already been an abort, we will need to query the leakage oracle again.

B.1 All-to-all Multibit Broadcast

We show how to edit the protocol $\mathsf{BC\text{-}FB}_p$ to implement all-to-all multibit broadcasts, meaning we can broadcast k multibit messages from k not-necessarily distinct parties in a single broadcast. The edited protocol leaks a fraction p of a bit in total. While this transformation is not essential to compile MPC protocols to topology-hiding ones, it will cut down the round complexity by a factor of n times the size of a message.

First observe that $\mathsf{BC\text{-}FB}_p$ actually works also to broadcast multiple bits. Instead of sending a single bit during the random-walk protocol, it is enough that parties send vectors of ciphertexts. That is, in each round parties send a vector $[\vec{b_1}, \ldots, \vec{b_\ell}, u]$.

Now we show how to achieve an all-to-all broadcast. Assume each party P_i wants to broadcast some k-bit message, $(b_1, \ldots, b_k)$. We consider a vector of length nk, where each of the n parties is assigned to k slots for k bits of its message. Each of the vectors $\vec{b_i}$ in the vector $[\vec{b_1}, \ldots, \vec{b_\ell}, u]$ described above will be of this form. P_i will use the slots from $n(i-1)$ to ni to communicate its message. This means that P_i will have as input vector $\vec{b_i} = (0, \ldots, 0, b_1, \ldots, b_k, 0, \ldots, 0)$. Then,

in the Aggregate Stage, the parties will input their input message into their corresponding slots (by homomorphically OR'ing the received vector with its input message). At the end of the protocol, each party will receive the output containing the broadcast message of each party P_j in the slots $n(j-1)$ to nj.

Lemma 3. *Protocol BC-FB$_p$ can be edited to an all-to-all multi-bit broadcast MultibitBC$_p$, which is secure against an adversary, who statically passively corrupts and adaptively crashes any number of parties and leaks at most a fraction p of a bit. The round complexity of MultibitBC$_p$ is the same as for BC-FB$_p$.*

Proof. This involves the following simple transformation of protocol BC-FB$_p$. Note that BC-FB$_p$ is already multibit; during the random-walk protocol, parties send around vectors of ciphertexts: $[\vec{b}, u] := [b_1, \ldots, b_\ell, u]$. In the transformed protocol we will substitute each ciphertext encrypting a bit b_i with a vector of ciphertexts of length m, containing encryptions of a vector of bits $\vec{b_i}$. That is, we now think of parties sending a vector of vectors $[\vec{b_1}, \ldots, \vec{b_\ell}, u]$. Technically, we "flatten" these vectors, that is, the parties will send vectors of length $m\ell + 1$ of ciphertexts.

Let us now explain the transformation. For an all-to-all broadcast, each party, P_i, wants to broadcast some k-bit message, $(b_1, \ldots, b_k)$. Consider a vector of ciphertexts of length nk, where each of the n parties is assigned to k slots for k bits of its message. Each of the vectors $\vec{b_i}$ in the vector $[\vec{b_1}, \ldots, \vec{b_\ell}, u]$ described above will be of this form. P_i will use the slots from $n(i-1)$ to ni to communicate its message.

We now have a look at the Aggregate Stage in the transformed protocol MultibitBC$_p$.

- Every party P_i who wants to send the k bit message $(b_1, \ldots, b_k)$ prepares its input vector $\vec{b_i} = (0, \ldots, 0, b_1, \ldots, b_k, 0, \ldots, 0)$ by placing the bits $b_1, \ldots, b_k$ in positions from $n(i-1)$ to ni.
- At the beginning of the Aggregate Stage, the recipient P_o with the input vector $\vec{b_o}$ sends the ciphertext $[\vec{b_o} \vee u_o, \vec{1}, \ldots, \vec{1}, u_o]_{\mathtt{pk}^{(1)}_{i\to 1}}$ to its first neighbor. All other ciphertexts to all other neighbors j are just $[\vec{1}, \ldots, \vec{1}, 1]_{\mathtt{pk}^{(1)}_{i\to j}}$ [7].

  Every other party P_i starts the protocol with sending the ciphertext tuple $[\vec{1}, \ldots, \vec{1}, 1]_{\mathtt{pk}^{(1)}_{i\to j}}$ to every neighbor j.
- Upon receiving a ciphertext at round r from a neighbor j, $[\vec{b_1}, \ldots, \vec{b_\ell}, u]_{pk^{(t)}_{j\to i}}$, party P_i takes its input vector $\vec{b_i}$ and homomorphically OR's the vector $(\vec{b_i} \vee u_i, \ldots, \vec{b_i} \vee u_i, u_i)$ containing ℓ copies of the vector $\vec{b_i} \vee u_i$ to the ciphertext. The result is sent along the walk.

The rest of the protocol MultibitBC$_p$ proceeds analogously to BC-FB$_p$.

A quick check of correctness tells us that when a message is not made unhappy, and starts with 0's in the appropriate places, every party's broadcast message eventually gets OR'd in a different spot in the message vector, and so every party will get that broadcast.

A quick check of soundness tells us that the simulator works just as before: it simulates with the encrypted output (all nk bits) when there was no abort, and with a query to the leakage function if there was one. □

B.2 Sequential Execution Without Aggregated Leakage

We show how to construct a protocol, which implements any number of sequential executions of the protocol MultibitBC$_p$, while preserving the leakage of a fraction p of a bit in total. The construction makes non-black-box use of the unhappy bits used in MultibitBC$_p$. The idea is simply to preserve the state of the unhappy bits between sequential executions. That is, once some party sees a crash, it will cause all subsequent executions to abort.

[7] We are abusing notation: $\vec{b_o} \vee u_o$ means that we OR u_i with every coordinate in $\vec{b}$.

Lemma 4. *There exists a protocol, which implements any number k of executions of the protocol $\mathsf{MultibitBC}_p$, is secure against an adversary, who statically passively corrupts and adaptively crashes any number of parties and leaks at most a fraction p of a bit in total. The complexity of the constructed protocol is k times the complexity of $\mathsf{MultibitBC}_p$.*

Proof. The construction makes non-black-box use of the unhappy bits used in $\mathsf{MultibitBC}_p$. The idea is simply to preserve the state of the unhappy bits between sequential executions. That is, once some party sees a crash, it will cause all subsequent executions to abort.

Correctness and complexity of the above construction are trivial, since it simply executes the protocol $\mathsf{MultibitBC}_p$ k times.

We now claim that any leakage happens only in the one execution of protocol $\mathsf{MultibitBC}_p$, in which the first crash occurs. Once we show this, it is easy to see that the constructed protocol executing $\mathsf{MultibitBC}_p$ k times leaks at most a fraction p of a bit.

By Theorem 3, any execution without crashes causes no leakage (it an be easily simulated as in the setting with only passive corruptions and no fail-stop adversary). Further, assume that any party P_c crashes before $\mathsf{BC\text{-}FB}_p$ starts. Let $\mathbf{N}_G(a)$ be all of P_a's neighbors; all of them will have their unhappy bit set to 1. Because of the correctness of the random-walk protocol embedded within $\mathsf{BC\text{-}FB}_p$, the random walk will hit every node in the connected component, and so is guaranteed to visit a node in $\mathbf{N}_G(a)$. Therefore, every walk will become a dummy walk, which is easily simulated. □

Remark 1. We note that the above technique to sequentially execute protocols which leak p bits and are secure *with abort* can be applied to a more general class of protocols (in particular, not only to our topology-hiding broadcast). The idea is that if a protocol satisfies the property that any abort before it begins implies that the protocol does not leak any information, then it can be executed sequentially leaking at most p bits.

B.3 Topology-Hiding Computation

We are now ready to compile any MPC protocol (secure against an adversary, who statically passively corrupts and adaptively crashes any number of parties) into one that is topology-hiding and leaks at most a fraction p of a bit.

To do this, it is enough to do a standard transformation using public key infrastructure. Let Π_{MPC} be a protocol that runs in M rounds. First, the parties use one all-to-all multi-bit topology-hiding broadcast protocol to send each public key to every other party. Then, each round of Π_{MPC} is simulated: the parties run n all-to-all multi-bit topology hiding broadcasts simultaneously to send the messages sent in that round encrypted under the corresponding public keys. After the broadcasts, each party can use their secret key to decrypt their corresponding messages.

Theorem 6. *Assume PKCR exists. Then, we can compile any MPC protocol Π_{MPC} that runs in M rounds into a topology-hiding protocol with leakage function $\mathcal{L}_{FB_p}$, that runs in $MR+1$ rounds, where R is the round complexity of $\mathsf{BC\text{-}FB}_p$.* [8]

Proof. Recall the generic transformation for taking UC-secure topology-hiding broadcast and compiling it into UC-secure topology-hiding MPC using a public key infrastructure. Every MPC protocol with M rounds, Π_{MPC}, has at each round each party sending possibly different messages to every other party. This is a total of $O(n^2)$ messages sent at each round, but we can simulate this with n separate multi-bit broadcasts.

To transform Π_{MPC} into a topology-hiding protocol in the fail-stop model, given a multi-bit topology-hiding broadcast, we do the following:

[8] In particular, the complexity of $\mathsf{BC\text{-}FB}_p$ is $n \cdot \rho \cdot 2\mathtt{T}$, where κ is the security parameter, $\tau = \log(n) + \kappa$, $\mathtt{T} = 8n^3\tau$ is the length of a walk and $\rho = \tau/(p' - 2^{-\tau})$ is the number of repetitions of a phase (with $p' = 1/\lfloor 1/p \rfloor$).

– Setup phase. The parties use one multi-bit topology-hiding broadcast to give their public key to every other party.
– Each round of Π_{MPC}. For each party P_i that needs to send a message of k bits to party P_j, P_i encrypts that message under P_j's public key. Then, each party P_i broadcasts the $n-1$ messages it would send in that round of Π_{MPC}, one for each $j \neq i$, encrypted under the appropriate public keys. That is, P_i is the source for one multi-bit broadcast. All these multi-bit broadcasts are simultaneously executed via an all-to-all multi-bit broadcast, where each party broadcast a message of size $(n-1)k$ times.
  After the broadcasts, each node can use their secret key to decrypt the messages that were for them and continue with the protocol.
– At the end of the protocol, each party now has the output it would have received from running Π_{MPC}, and can compute its respective output.

First, this is a correct construction. We will prove this by inducting on the rounds of Π_{MPC}. To start, all nodes have all information they would have had at the beginning of Π_{MPC} as well as public keys for all other parties and their own secret key. Assume that the graph has just simulated round $r-1$ of Π_{MPC} and each party has the information it would have had at the end of round $r-1$ of Π_{MPC} (as well as the public keys etc). At the end of the r'th simulated round, each party P_i gets encryptions of messages sent from every other party P_j encrypted under P_i's public key. These messages were all computed correctly according to Π_{MPC} because all other parties had the required information by the inductive hypothesis. P_i can then decrypt those messages to get the information it needs to run the next round. So, by the end of simulating all rounds of Π_{MPC}, each party has the information it needs to complete the protocol and get its respective output.

Security of this construction (and, in particular, the fact that it only leaks a fraction p of a bit) follows directly from Lemma 3 and Lemma 4. □

We can now conclude that any MPC functionality can be implemented by a topology-hiding protocol. Since PKCR is implied by either DDH, QR or LWE, we get the following theorem as a corollary.

Theorem 1. *If DDH, QR or LWE is hard, then any MPC functionality $\mathcal{F}$ can be realized by a topology-hiding protocol which is secure against an adversary that does any number of static passive corruptions and adaptive crashes, leaking an arbitrarily small fraction p of a bit. The round and communication complexity is polynomial in κ and $1/p$.*

Proof. Because every poly-time computable functionality $\mathcal{F}$ has an MPC protocol [CLOS02], we get that Theorem 6 implies we can get topology-hiding computation. The round and communication complexity is implied by Theorem 6 and the complexity of $\mathsf{MultibitBC}_p$. □

C Deeply Fully-Homomorphic Public-Key Encryption

In this section we present the formal definition of deeply fully-homomorphic public-key encryption from Section 5.1.

Our protocol requires a PKE scheme $\mathcal{E}$ where (a) one can add and remove layers of encryption, while (b) one can homomorphically compute any function on encrypted bits (independent of the number of layers). This will be captured by three additional algorithms: $\mathsf{AddLayer}_r$, $\mathsf{DelLayer}_r$, and HomOp_r, operating on ciphertexts with r layers of encryption (we will call such ciphertexts level-r ciphertexts). A level-r ciphertext is encrypted under a level-r public key (we assume that each level can have different key space).

Definition 2. *A* deeply fully-homomorphic public-key encryption *(DFH-PKE) scheme is a PKE scheme with additional algorithms $\mathsf{AddLayer}_r$, $\mathsf{DelLayer}_r$, and HomOp_r. We define additional public-key spaces $\mathcal{PK}_r$ and ciphertext spaces $\mathcal{C}_r$, for public keys and ciphertexts on level r. We require that $\mathcal{PK}_1 = \mathcal{PK}$ and $\mathcal{C}_1 = \mathcal{C}$. Let $\mathcal{F}$ be the family of efficiently computable functions.*

– *The algorithm* $\mathsf{AddLayer}_r : \mathcal{C}_r^* \times \mathcal{PK}_r \times \mathcal{SK} \to \mathcal{C}_{r+1}^* \times \mathcal{PK}_{r+1}$ *takes as input a level-r ciphertext* $[\![m]\!]_{\mathbf{pk}}$*, the corresponding level-r public key* $\mathbf{pk}$*, and a new secret key* sk*. It outputs a level-*$(r+1)$ *ciphertext and the level-*$(r+1)$ *public key, under which it is encrypted.*
– *The algorithm* $\mathsf{DelLayer}_r : \mathcal{C}_{r+1}^* \times \mathcal{PK}_{r+1} \times \mathcal{SK} \to \mathcal{C}_r^* \times \mathcal{PK}_r$ *deletes a layer from a level-*$(r+1)$ *ciphertext.*
– *The algorithm* $\mathsf{HomOp}_r : \mathcal{C}_r^* \times \mathcal{PK}_r \times \mathcal{F} \to \mathcal{C}_r$ *takes as input some* k *level-r ciphertexts encrypted under the same level-r public key, the corresponding public key, and a* k*-ary function* f*. It outputs a level-r ciphertext that contains* f *of the encrypted messages.*

For convenience, it will be easy to describe the security of our enhanced encryption scheme with the help of an algorithm $\mathsf{Leveled\text{-}Encrypt}_r$, which takes as input a vector of plain messages and a level-r public key, and outputs a vector of level-r ciphertexts[9].

Definition 3. *For a DFH-PKE scheme, we additionally define the algorithm* $\mathsf{Leveled\text{-}Encrypt}_r : \mathcal{M}^* \times \mathcal{PK}_r \to \mathcal{C}_r^* \times \mathcal{PK}_r$ *that outputs the level-r encryptions of the messages* $\vec{m}$ *and the corresponding level-r public key.*

Intuitively, we will require that from the output of $\mathsf{AddLayer}_r$ ($\mathsf{DelLayer}_r$) one cannot obtain any information on the underlying layers of encryption. That is, that the output of $\mathsf{AddLayer}_r$ ($\mathsf{DelLayer}_r$) is indistinguishable from a level-$(r+1)$ (level-r) encryption of the message. We will also require that the output of HomOp_r is indistinguishable from a level-r encryption of the output of the functions applied to the messages.

Definition 4. *We require that a DFH-PKE scheme satisfies the following properties:*

Aggregate Soundness. *For every* r*, every vector of messages* $\vec{m}$ *and every efficiently computable pair of level-r public keys* $\mathbf{pk_1}$ *and* $\mathbf{pk_2}$*,*

$$\{\mathsf{AddLayer}_r([\![\vec{m}]\!]_{\mathbf{pk_1}}, \mathbf{pk_1}, \mathit{sk}; U^*) : (\mathit{pk}, \mathit{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa; U^*)\}$$

$$\approx_c$$

$$\left\{(\mathsf{Leveled\text{-}Encrypt}_{r+1}(\vec{m}, \mathbf{pk_2'}; U^*), \mathbf{pk_2'}) : \begin{array}{c}(\mathit{pk}, \mathit{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa; U^*),\\ ([\![0]\!]_{\mathbf{pk_2'}}, \mathbf{pk_2'}) \leftarrow \mathsf{AddLayer}_r([\![0]\!]_{\mathbf{pk_2}}, \mathbf{pk_2}, \mathit{sk}; U^*)\end{array}\right\}$$

Decrypt Soundness. *For every* r*, every vector* $\vec{m}$ *and every efficiently computable level-r public key* $\mathbf{pk_1}$*,*

$$\left\{\mathsf{DelLayer}_r([\![\vec{m}]\!]_{\mathbf{pk}}, \mathbf{pk}, \mathit{sk}; U^*) : \begin{array}{c}(\mathit{pk}, \mathit{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa; U^*),\\ ([\![0]\!]_{\mathbf{pk}}, \mathbf{pk}) \leftarrow \mathsf{AddLayer}_r([\![0]\!]_{\mathbf{pk_1}}, \mathbf{pk_1}, \mathit{sk}; U^*)\end{array}\right\}$$

$$\approx_c$$

$$\{(\mathsf{Leveled\text{-}Encrypt}_r(\vec{m}, \mathbf{pk_1}; U^*), \mathbf{pk_1})\}$$

Full-Homomorphism. *For every vector of messages* $\vec{m} \in \mathcal{M}^*$*, every level-r public key* $\mathbf{pk}$*, every vector of ciphertexts* $\vec{c} \in \mathcal{C}^*$ *and every function* $f \in \mathcal{F}$ *where* $\mathsf{Leveled\text{-}Encrypt}_r(\vec{m}, \mathbf{pk}) = \vec{c}$*,*

$$\{(\vec{m}, \vec{c}, \mathbf{pk}, f, \mathsf{Leveled\text{-}Encrypt}_r(f(\vec{m}), \mathbf{pk}; U^*))\}$$

$$\approx_c$$

$$\{(\vec{m}, \vec{c}, \mathbf{pk}, f, \mathsf{HomOp}_r(\vec{c}, \mathbf{pk}, f; U^*))\}$$

[9] This algorithm can be obtained by keeping an encryption of 0 and 1 as part of the leveled public key and rerandomizing the ciphertext using HomOp_r.

Note that $\mathsf{AddLayer}_r$ and $\mathsf{DelLayer}_r$ produce both the level-r encrypted messages and the level-r public key. In the case where we only need the public key, we will just call the procedure $\mathsf{AddLayer}_r([\![0]\!]_{\mathbf{pk}}, \mathbf{pk}, \mathtt{sk})$, since the encrypted message does not matter for producing a new public key — the same applies for $\mathsf{DelLayer}_r$.

Also note that one can create a level-r public key generating r level-1 key pairs $(\mathtt{pk}_i, \mathtt{sk}_i) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and using $\mathsf{AddLayer}$ to add the public keys one by one. Furthermore, with all secret keys $(\mathtt{sk}_1, \ldots, \mathtt{sk}_r)$ used in the creation of some level-r public key $\mathbf{pk}$, we can define a combined level-r secret key $\mathbf{sk} = (\mathtt{sk}_1, \ldots, \mathtt{sk}_r)$, which we can use to decrypt a level-r ciphertext by calling $\mathsf{DelLayer}$ r times.

C.1 Instantiation of DFH-PKE from FHE

We show how to instantiate DFH-PKE from FHE. As required from the DFH-PKE scheme, the level-1 public key space and ciphertext space are the FHE public key space and FHE ciphertext space respectively, i.e., $\mathcal{PK}_1 = \mathcal{PK}$ and $\mathcal{C}_1 = \mathcal{C}$. For $r > 1$, a level-r public key and ciphertext spaces are $\mathcal{PK}_r = \mathcal{PK} \times \mathcal{C}$ and $\mathcal{C}_r = \mathcal{C}$, respectively.

Notation. We denote by $\mathrm{FHE}.\mathsf{Encrypt}(m, \mathtt{pk})$ the FHE encryption algorithm that takes message m and encrypts under public key $\mathtt{pk}$. In the same way, the FHE decryption algorithm is denoted by $\mathrm{FHE}.\mathsf{Decrypt}$. The FHE evaluation algorithm is defined as

$$\mathrm{FHE}.\mathsf{HomOp}([m_1, \ldots, m_n]_{\mathtt{pk}}, \mathtt{pk}, f) \coloneqq [f(m_1, \ldots, m_n)]_{\mathtt{pk}}.$$

It gets as input a vector of encrypted messages under $\mathtt{pk}$, the public key $\mathtt{pk}$ and the function to evaluate, and it returns the output of f applied to the messages.

In the following we define the algorithms to add and remove layers:

Algorithm $\mathsf{AddLayer}_r((\mathtt{c}_1, \ldots, \mathtt{c}_n), \mathbf{pk}, \mathtt{sk})$

Let $\mathtt{pk}$ be the corresponding public key of $\mathtt{sk}$.
$\mathtt{c}'_i \leftarrow \mathrm{FHE}.\mathsf{Encrypt}(\mathtt{c}_i, \mathtt{pk})$.
$\mathbf{pk}' \leftarrow (\mathtt{pk}, \mathrm{FHE}.\mathsf{Encrypt}(\mathbf{pk}, \mathtt{pk}))$.
return $((\mathtt{c}'_1, \ldots, \mathtt{c}'_n), \mathbf{pk}')$.

Algorithm $\mathsf{DelLayer}_r((\mathtt{c}'_1, \ldots, \mathtt{c}'_n), \mathbf{pk}', \mathtt{sk})$

Parse $\mathbf{pk}' = (\mathtt{pk}, [\mathbf{pk}]_{\mathtt{pk}})$.
$\mathbf{pk} \leftarrow \mathrm{FHE}.\mathsf{Decrypt}([\mathbf{pk}]_{\mathtt{pk}}, \mathtt{sk})$.
$\mathtt{c}_i \leftarrow \mathrm{FHE}.\mathsf{Decrypt}(\mathtt{c}'_i, \mathtt{sk})$.
return $((\mathtt{c}_1, \ldots, \mathtt{c}_n), \mathbf{pk})$.

Notice the recursive nature of leveling; to make notation less cumbersome, let $\mathbf{pk}_r = (\mathtt{pk}_r, [\mathtt{pk}_{r-1}, [\ldots [\mathtt{pk}_1]_{\mathtt{pk}_2} \ldots]_{\mathtt{pk}_{r-1}}]_{\mathtt{pk}_r})$, and $[\![m]\!]_{\mathbf{pk}_r}$ denotes the leveled ciphertext, i.e., $[\![m]\!]_{\mathbf{pk}_r} = [[\ldots [m]_{\mathtt{pk}_1} \ldots]_{\mathtt{pk}_{r-1}}]_{\mathtt{pk}_r}$. Hence, it is easy to see that the two algorithms above accomplish the following:

$$\mathsf{AddLayer}_r([\![\vec{m}]\!]_{\mathbf{pk}_r}, \mathbf{pk}_r, \mathtt{sk}_{r+1}) = ([\![\vec{m}]\!]_{\mathbf{pk}_{r+1}}, \mathbf{pk}_{r+1})$$

and

$$\mathsf{DelLayer}_r([\![\vec{m}]\!]_{\mathbf{pk}_{r+1}}, \mathbf{pk}_{r+1}, \mathtt{sk}_r) = ([\![\vec{m}]\!]_{\mathbf{pk}_r}, \mathbf{pk}_r)$$

In the following, we show how to apply any function f on any vector of level-r ciphertexts. It is clear that if the ciphertexts are level-1 ciphertexts, we can apply f using FHE directly. If the ciphertexts are level-r ciphertexts for $r > 1$, we FHE evaluate the ciphertexts and public key with a recursive function call on the previous level. More concretely, we use the following recursive algorithm to apply f to any vector of level-r ciphertexts:

Algorithm $\mathsf{HomOp}_r((\mathsf{c}_1, \ldots, \mathsf{c}_n), \mathbf{pk}, f)$

if $r = 1$ **then**
  Parse $\mathbf{pk} = \mathtt{pk}$.
  return $\mathrm{FHE.HomOp}((\mathsf{c}_1, \ldots, \mathsf{c}_n), \mathtt{pk}, f)$.
end if
Parse $\mathbf{pk} = (\mathtt{pk}, [\mathbf{pk}_{r-1}]_{\mathtt{pk}})$, $\mathsf{c}_i = [\mathsf{c}'_i]_{\mathtt{pk}}$.
Let $f'(\cdot,\cdot) \coloneqq \mathsf{HomOp}_{r-1}(\cdot,\cdot,f)$.
return $\mathrm{FHE.HomOp}(([(\mathsf{c}'_1, \ldots, \mathsf{c}'_n)]_{\mathtt{pk}}, [\mathbf{pk}_{r-1}]_{\mathtt{pk}}), \mathtt{pk}, f')$.

Lemma 5. *For any r, algorithm HomOp_r is correct on leveled ciphertexts.*

Proof. We want to show that for a vector of level-r ciphertexts $\vec{c} = [\![\vec{m}]\!]_{\mathbf{pk}}$, $\mathsf{HomOp}_r(c, \mathbf{pk}, f) = [\![f(\vec{m})]\!]_{\mathbf{pk}}$. We will prove this via induction on r.

For the base case, consider $r = 1$. Here we go into the if statement, and the algorithm returns $\mathrm{FHE.HomOp}([\vec{m}]_{\mathtt{pk}}, \mathtt{pk}, f) = [f(\vec{m})]_{\mathtt{pk}}$ by the correctness of the FHE scheme.

Now, assume that $\mathsf{HomOp}_{r-1}([\![\vec{m}]\!]_{\mathbf{pk}_{r-1}}, \mathbf{pk}_{r-1}, f) = [\![f(\vec{m})]\!]_{\mathbf{pk}_{r-1}}$ for all messages $\vec{m}$ encrypted under $r-1$ levels of keys. Calling HomOp_r on $[\![\vec{m}]\!]_{\mathbf{pk}_r}$ results in returning

$$\begin{aligned}
&\mathrm{FHE.HomOp}(([\![\vec{m}]\!]_{\mathbf{pk}_r}, [\mathbf{pk}_{r-1}]_{\mathtt{pk}_r}), \mathtt{pk}_r, \mathsf{HomOp}_{r-1}(\cdot,\cdot,f)) \\
&= [\mathsf{HomOp}_{r-1}([\![\vec{m}]\!]_{\mathbf{pk}_{r-1}}, \mathbf{pk}_{r-1}, f)]_{\mathtt{pk}_r} \\
&= [\![f(\vec{m})]\!]_{\mathbf{pk}_r}
\end{aligned}$$

by correctness of the FHE homomorphic evaluation.

We are also able to encrypt in a leveled way by exploiting the fully-homomorphic properties of the scheme, using the FHE.HomOp algorithm to apply encryption.

Algorithm $\mathsf{Leveled\text{-}Encrypt}_r(\vec{m}, \mathbf{pk})$

if $r = 1$ **then**
  Parse $\mathbf{pk} = \mathtt{pk}$
  return $(\mathrm{FHE.Encrypt}(m_i, \mathtt{pk}))_i$
end if
Parse $\mathbf{pk} = (\mathtt{pk}, [\mathbf{pk}_{r-1}]_{\mathtt{pk}})$.
Let $[\vec{m}]_{\mathtt{pk}} = (\mathrm{FHE.Encrypt}(m_i, \mathtt{pk}))_i$.
return $\mathrm{FHE.HomOp}(([\vec{m}]_{\mathtt{pk}}, [\mathbf{pk}_{r-1}]_{\mathtt{pk}}), \mathtt{pk}, \mathsf{Leveled\text{-}Encrypt}_{r-1})$

Finally, we need to prove that adding a fresh layer is equivalent to looking like a fresh random encryption.

Lemma 6. $\mathsf{Leveled\text{-}Encrypt}_r(\vec{m}, \mathbf{pk}_r) = [\![\vec{m}]\!]_{\mathbf{pk}_r}$.

Proof. We will prove this by induction on r. For $r = 1$, it follows from the base case that

$$\mathsf{Leveled\text{-}Encrypt}_1(\vec{m}, \mathbf{pk}_1) = \mathrm{FHE.Encrypt}(\vec{m}, \mathtt{pk}_1) = [\![\vec{m}]\!]_{\mathbf{pk}_1}.$$

Now, assume that for $r-1$, $\mathsf{Leveled\text{-}Encrypt}_{r-1}(\vec{m}, \mathbf{pk}_{r-1}) = [\![\vec{m}]\!]_{\mathbf{pk}_{r-1}}$. This means that when we call $\mathsf{Leveled\text{-}Encrypt}_r(\vec{m}, \mathbf{pk}_r)$, we return

$$\mathrm{FHE.HomOp}([\vec{m}]_{\mathtt{pk}_r}, [\mathbf{pk}_{r-1}]_{\mathtt{pk}_r}, \mathtt{pk}_r, \mathsf{Leveled\text{-}Encrypt}_{r-1}) = [[\![\vec{m}]\!]_{\mathbf{pk}_{r-1}}]_{pk_r} = [\![\vec{m}]\!]_{\mathbf{pk}_r}$$

as desired.

Lemma 7. *The instantiation of DFH-PKE from FHE presented above satisfies the properties Aggregate Soundness, Decrypt Soundness and Full-Homomorphism, presented in Definition 4.*

Proof. **Aggregate Soundness.** The algorithm AddLayer returns a tuple $(\llbracket \vec{m} \rrbracket_{\mathbf{pk}}, \mathbf{pk}')$, where $\vec{m}$ is a vector of messages, and $\mathbf{pk}' = (\mathtt{pk}, \text{FHE.}\mathsf{Encrypt}(\mathbf{pk}, \mathtt{pk}))$ is a pair containing a fresh public key pk and an encryption of a level-r key $\mathbf{pk}$ under the fresh public key pk. Observe that this is exactly a level-$(r+1)$ key.

The tuple that consists of $(\mathsf{Leveled\text{-}Encrypt}_{r+1}(\vec{m}, \mathbf{pk_1}; U^*), \mathbf{pk_1})$, where $\mathbf{pk_1}$ is a level-$(r+1)$ public key obtained from adding a fresh layer to a level-r public key, has the same distribution: the first part of both tuples contain fresh FHE encryptions of level-r ciphertexts, and the second part is a level-$(r+1)$ public key.

Decrypt Soundness. This property is trivially achieved given the correctness of the FHE decryption algorithm and $\mathsf{Leveled\text{-}Encrypt}_r$.

Full-Homomorphism. The $\mathsf{Leveled\text{-}Encrypt}_r$ algorithm returns a level-r encryption of $f(\vec{m})$ which is the result of applying FHE homomorphic operations on a level-r ciphertext. The algorithm HomOp_r also returns a level-r ciphertext output by the FHE homomorphic operation.

D Topology-Hiding Computation from DFH-PKE

In this section, we present a detailed description of protocol DFH-THC from Section 5.2.

We will use DFH-PKE to alter the RandomWalkPhase protocol (and by extension we can alter $\mathsf{ProbabilisticRandomWalkPhase}_p$). Then, executing protocols BC-OB and $\mathsf{BC\text{-}FB}_p$ that leak one bit and a fraction of a bit respectively will be able to evaluate any poly-time function instead, while still leaking the same amount of information as a broadcast using these random walk protocols. The concept is simple. During the Aggregate Stage, parties will add a leveled encryption of their input and identifying information to a vector of ciphertexts, while adding a layer — we will not need sequential id's if each party knows where their input should go in the function. Then, at the end of the Aggregate Stage, nodes homomorphically evaluate f', which is the composition of a parsing function, to get one of each input in the right place, and f, to evaluate the function on the parsed inputs. The result is a leveled ciphertext of the output of f. This ciphertext is un-layered in the Decrypt Stage so that by the end, the relevant parties get the output.

For completeness, we give a detailed description of the modified protocol RandomWalkPhase leaking one bit, which we call DFH-RandomWalkPhase:

Initialization Stage. Each party P_i has its own input bit b_i and unhappiness bit u_i. Each party P_i knows the function f on n variables that the graph wants to compute, and generates $\mathtt{T} \cdot d_i$ keypairs and $\mathtt{T} - 1$ permutations on d_i elements (d_i is the number of neighbors for party i). P_i also generates a unique ID (or uses a given sequential or other ID) p_i. If party P_i witnessed an abort from the last phase, it becomes unhappy, setting its unhappy bit $u_i = 1$.

Aggregate Stage. *Round 1.* Each party P_i sends to each neighbor P_j a vector of level-1 ciphertexts under $\mathtt{pk}_{i\to j}^{(1)}$ containing the input bit b_i, id p_i, unhappy bit u_i and a bit v_i indicating whether the walk is dummy or not.

If P_i is the party that gets the output in that phase, i.e., $P_i = P_o$, then it sends to the first neighbor an encryption of b_i, p_i, u_i and a bit $v_i = 0$ indicating that the walk should not be dummy. To all other neighbors, $v_i = 1$. In the case where $P_i \neq P_o$, $v_i = 1$ as well.

Round $r \in [2, T]$. Let $k = \pi_i^{(r)}(j)$. Upon receiving a vector of level-$(r-1)$ ciphertexts from P_j. Party P_i uses $\mathtt{sk}_{i\to k}^{(r)}$ to add a fresh layer with AddLayer to the vector of ciphertexts. The function AddLayer will return the vector $\vec{c}$ of level-r ciphertexts with the corresponding level-r public key $\mathbf{pk}$. Then, P_i will encrypt its own input, id and unhappybit via Leveled-Encrypt under $\mathbf{pk}$ and appends these ciphertexts to $\vec{c}$. It then sends to P_k the level-r public key and all the level-r ciphertexts.

If no vector of ciphertexts was received from P_j (i.e. P_j aborted), P_i generates a fresh level-r public key $\mathbf{pk}$ and secret key $\mathbf{sk}$. It then generates a vector of level-r ciphertexts containing

the bit 1 using Leveled-Encrypt under **pk**. The size of this vector corresponds to the size of the vector containing the dummy bit, r input bits, r ids, and r unhappy bits.

Evaluation. We are now at the last step in the walk. If P_i received an encrypted vector of level-T ciphertexts from P_j, it evaluates the vector using HomOp_T on the function f' which does the following: if the dummy bit is 1 or any unhappy bit set to 1, the function evaluates to $\perp$. Otherwise, it arranges the inputs by ids and evaluates f on the arranged inputs. That is, it evaluates $f \circ \mathsf{parse}$, where $\mathsf{parse}((m_{i_1}, p_{i_1}), \ldots, (m_{i_T}, p_{i_T})) = (m_1, \ldots, m_n)$. More concretely, for the vector of ciphertexts $\vec{c}$ and level-T public key **pk** received from P_j, P_i evaluates $\hat{c} \leftarrow \mathsf{HomOp}(\vec{c}, \mathbf{pk}, f')$, and sends $\hat{c}$ to P_j.

If P_i did not receive a message from P_j, or u_i has been set to 1, P_i sends a ciphertext containing $\perp$: it generates a fresh level-T public key **pk** and secret key **sk**, and uses Leveled-Encrypt under **pk** to send to P_j a level-T ciphertext containing $\perp$.

Decrypt Stage. *Round* $r \in [T, 2]$ If P_i receives a level-r ciphertext $\mathtt{c}$ from P_j, party P_i will delete a layer using the secret key $\mathtt{sk}_{i\to j}^{(r)}$ that was used to add a layer of encryption at round r of the Aggregate Stage. Otherwise, it uses Leveled-Encrypt to encrypt the message $\perp$ under the level-$(r-1)$ public key that was received in round r during the Aggregate Stage.

Output. If P_i is the party that gets the output in that phase, i.e., $P_i = P_o$ and it receives a level-1 ciphertext $\mathtt{c}$ from its first neighbour, P_i computes the output message using Decrypt using the secret key $\mathtt{sk}_{i\to 1}^{(1)}$. In any other case, P_i outputs $\perp$. P_i also outputs its unhappy bit u_i.

Now, DFH-THC runs the protocol DFH-RandomWalkPhase n times, similarly to BC-OB.

Protocol $\mathsf{DFH\text{-}THC}(\mathtt{T}, (d_i, \mathsf{input}_i)_{P_i \in \mathcal{P}}, f)$

Each party P_i sets $\mathsf{output}_i = \mathbf{1}$ and $u_i = 0$.
for o from 1 to n **do**
  Parties jointly execute $\left((\mathsf{input}_i^{temp}, u_i^{temp})_{P_i \in \mathcal{P}}\right) =$
    $\mathsf{DFH\text{-}RandomWalkPhase}(\mathtt{T}, P_o, (d_i, \mathsf{input}_i, u_i)_{P_i \in \mathcal{P}}, f)$.
  Party P_o sets $\mathsf{output}_o = \mathsf{input}_o^{temp}$.
  Each party P_i sets $u_i = u_i^{temp} \vee u_i$.
end for
Each party P_i **outputs** output_i if $\mathsf{output}_i \neq \perp$.

Theorem 4. *For security parameter* κ, $\tau = \log(n) + \kappa$, $T = 8n^3\tau$, *and* $\rho = \tau/(p' - 2^{-\tau})$, *where* $p' = 1/\lfloor 1/p \rfloor$, *the protocol* $\mathsf{DFH\text{-}THC}(T, \rho, (d_i, \mathsf{input}_i)_{P_i \in \mathcal{P}}))$ *topology-hidingly evaluates any poly-time function* f, $\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{FB_p}} || f$ *in the* $\mathcal{F}_{\text{NET}}$ *hybrid-world.*

Proof (Sketch). This proof will look almost exactly like the proof of Theorem 3. The simulator and its use of the leakage oracle will behave in nearly the same manner as before.

- During the Aggregate Stage, the simulator sends leveled encryptions of 1 of the appropriate size with the appropriate number of layers.
- During the Decrypt Stage, the simulator sends the output encrypted with the appropriate leveled keys.

Because $\mathsf{Leveled\text{-}Encrypt}_r$ is able to produce a distribution of ciphertexts that looks identical to $\mathsf{AddLayer}_r$, and by semantic security of the FHE scheme, no party can tell what other public keys were used except the most recently added one, the simulated ciphertexts and public keys are computationally indistinguishable from those in the real walk.

It is also worth pointing out that as long as the FHE scheme only incurs additive blowup in error and size, and $T = \mathsf{poly}(\kappa)$, the ciphertexts being passed around are only $\mathsf{poly}(\kappa)$ in size. □