Topology Hiding Computation on All Graphs

by

Rio LaVigne

B.S. Stanford University 2015

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2017

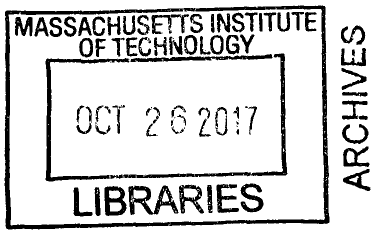
Author Signature redacted .....
Department of Electrical Engineering and Computer Science
August 31, 2017

Signature redacted

Certified by .. ..
Vinod Vaikuntanathan
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Signature redacted

Accepted by..... .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair of the Committee of Graduate Students



Topology Hiding Computation on All Graphs

by

Rio LaVigne

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2017, in partial fulfillment of the
requirements for the degree of
Masters of Science in Computer Science and Engineering

Abstract

A distributed computation in which nodes are connected by a partial communication graph is called *topology-hiding* if it does not reveal information about the graph beyond what is revealed by the output of the function. Previous results have shown that topology-hiding computation protocols exist for graphs of constant degree and logarithmic diameter in the number of nodes [Moran-Orlov-Richelson, TCC'15; Hirt et al., Crypto'16] as well as for other graph families, such as cycles, trees, and low circumference graphs [Akavia-Moran, Eurocrypt'17], but the feasibility question for general graphs was open.

In this work we positively resolve the above open problem: we prove that topology-hiding computation is feasible for *all* graphs under the Decisional Diffie-Hellman assumption.

Our techniques employ random or deterministic walks to generate paths covering the graph, upon which we apply the Akavia-Moran topology-hiding broadcast for chain-graphs (paths). To prevent topology information revealed by the random-walk, we design multiple graph-covering sequences that, together, are locally identical to receiving at each round a message from each neighbor and sending back a processed message from some neighbor (in a randomly permuted order).

Thesis Supervisor: Vinod Vaikuntanathan
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

First, I would like to thank my advisor Vinod Vaikuntanathan for guiding me through my research, and Shafi Goldwasser for giving me the opportunity to go to Israel to study; without going to Israel, I would have never met my collaborators. Next, I would like to acknowledge these collaborators, Tal Moran at IDC in Herzliya and Adi Akavia at The Academic College of Tel-Aviv Jaffa, for including me on this project. The time working with them was invaluable to this thesis. Finally, I would like ot acknowledge Luke Schaffer at MIT for pointing out that Universal Exploration Sequences could be used to derandomize some parts of our protocol.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The beautiful theory of secure multiparty computation (MPC) enables multiple parties to compute an arbitrary function of their inputs without revealing anything but the function's output [26, 11, 13]. In the original definitions and constructions of MPC, the participants were connected by a full communication graph (a broadcast channel and/or point-to-point channels between every pair of parties). In real-world settings, however, the actual communication graph between parties is usually not complete, and parties may be able to communicate directly with only a subset of the other parties. There is a lot of work, starting with Dolev et al.in 1993 [8], generalizing multiparty computation to these incomplete settings. However, these works all assume that the communication topology is public, or at least does not need to be hidden. In some cases, however, the graph itself is sensitive information (e.g., if you communicate directly only with your friends in a social network).

A natural question is whether we can successfully perform a joint computation over a partial communication graph while revealing no (or very little) information about the graph itself. In the information-theoretic setting, in which a variant of this question was studied by Hinkelman and Jakoby [17], the answer is mostly negative. The situation is better in the computational setting. Moran, Orlov and Richelson showed that topology-hiding computation *is* possible against static, semi-honest adversaries [22]; followed by constructions with improved efficiency that make only black-box use of underlying primitives [18]. However, all these protocol are restricted to communication graphs with *small diameter*. Specifically, these protocols address networks with diameter $D = O(\log n)$, logarithmic in the number

of nodes n (where the diameter is the maximal distance between two nodes in the graph). Akavia and Moran [1] showed that topology hiding computation is feasible also for *large diameter* networks of certain forms, most notably, cycles, trees, and low circumference graphs.

However, there are natural network topologies not addressed by the above protocols [22, 18, 1]. They include, for example, wireless and ad-hoc sensor networks (e.g. mesh networks for cellphones, etc), as in [9, 24]. The topology in these graphs is modeled by random geometric graphs [23], where, with high probability, the diameter and the circumference are simultaneously large [10, 2]. These qualities exclude the use of all aforementioned protocols. So, the question remained:

Is topology hiding MPC feasible for every network topology?

1.1 Our Results

In this work we prove that topology hiding MPC is feasible for *every* network topology under the Decisional Diffie-Hellman (DDH) assumption, thus positively resolving the above open problem. The adversary is static and semi-honest as in the prior works [22, 18, 1].[1] Our protocol also fits a stronger definition of security than that from prior works: instead of allowing the adversary to know who his neighbors are, he only gets pseudonyms; importantly, an adversary cannot tell if two nodes he controls share an honest neighbor.

Theorem 1.1.1 (Topology-hiding broadcast for all network topologies – informal)**.** *There exists a topology-hiding protocol realizing the broadcast functionality on every network topology (under DDH assumption and provided the parties are given an upper-bound n on the number of nodes).*

The formal theorem is stated and proved in chapter 5.

As in [22, 18, 1], given a *topology-hiding broadcast* for a point-to-point channels network, we can execute on top of it any MPC protocol from the literature that is designed for networks with broadcast channels; the resulting protocol remains topology-hiding.

[1] Moran et al.[22] consider also a fail-stop adversary for proving an impossibility result.

Theorem 1.1.2 (Compiling THC from THB and PKE – Informal). *Given topology hiding broadcast and public key encryption, then for any PPT protocol* Π *that is a secure multi-party protocol for a function* $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$*, there exists a PPT protocol* Π' *that is a topology-hiding protocol for a function* f.

Our MPC model is in the synchronous setting and is secure against a static, passive adversary that can corrupt an arbitrary number nodes. This theorem, theorem 4.2.1, and MPC model is formally stated and proved in chapter 4.

Since we have the existence of secure MPC in this model for all efficiently computable functionalities, [26, 11, 13], we conclude that *topology-hiding MPC* exists for all efficiently computable functionality and all network topologies.

1.2 High-Level Overview of Our Techniques

Our main innovation is the use of locally computable exploration sequences – walks, deterministic or random, that traverse the graph. We use these sequences to specify a path, view this path as a chain-graph, and then employ the topology-hiding broadcast protocol for chains of Akavia and Moran [1]. We discuss two methods for getting these sequences: random walks and universal exploration sequences. In this overview, we will describe how our protocol works with respect to *random-walks*. Extending these ideas to other kinds of sequences follows naturally.

A challenge we face is that the walk itself may reveal topology information. For example, a party can deduce the graph commute-time from the number of rounds before a returning visit by the walk. We therefore hide the random-walk by using multiple simultaneous random-walks (details below). The combination of all our random-walks obeys a simple communication structure: at every round each node receives an incoming message from each of its neighbors, randomly permutes the messages, and sends them back, one along each outgoing edge.

To give more details on our protocol, let us first recall the Akavia-Moran protocol for chain-graphs. The Akavia-Moran protocol proceeds in two phases: a forward and a backward phase. In the forward phase, messages are passed forward on the chain, where each

node adds its own encryption layer, and computes the OR of the received message with its bit using homomorphic multiplication (with proper re-randomizing). In the backward phase, the messages are passed backward along the same path, where each node deletes its encryption layer. At the end of the protocol, the starting node receives the plaintext value for the OR of all input bits. This protocol is augmented to run n instances simultaneously; each node initiates an execution of the protocol while playing the role of the first node. So, by the end of the protocol, each node has the OR of all bits, which will be equal to the broadcast bit. Intuitively, this achieves topology-hiding because at each step, every node receives an encrypted message and public key. An encryption of zero is indistinguishable from an encryption of 1, and so each node's view is indistinguishable from every other view.

We next elaborate on how we define our multiple random walks, focusing on two viewpoints: the viewpoint of a node, and the viewpoint of a message. We use the former to argue security, and the latter to argue correctness.

From the point of view of a node v with d neighbors, the random walks on the forward-phase are specified by choosing a sequence of independent random permutations $\pi_t : [d] \rightarrow [d]$, where in each forward-phase round t, the node forwards messages received from neighbor i to neighbor $\pi_t(i)$ (after appropriate processing of the message, as discussed above). The backward-phase follows the reverse path, sending incoming message from neighbor j to neighbor $i = \pi_t^{-1}(j)$, where t is the corresponding round in the forward-phase. Furthermore, recall that all messages are encrypted under semantically-secure encryption. This fixed communication pattern together with the semantic security of the messages content leads to the topology-hiding property of our protocol.

From the point of view of a message, at each round of the forward-phase the message is sent to a uniformly random neighbor. Thus, the path the message goes through is a random-walk on the graph.[2] A sufficiently long random walk covers the entire graph with overwhelming probability. In this case, the output is the OR of the inputs bits of *all* graph nodes, and correctness is guaranteed.

We can remove the randomness, and thus ensure our walks all traverse the graph, by

[2]We remark that the multiple random-walks are not independent; we take this into account in our analysis.

using Universal Exploration Sequences instead of random walks. These sequences are locally computable by each node and only require knowing how many nodes are in the network.

1.3 Related Work

Topology Hiding in Computational Settings. Table 1.1 compares our results to the previous results on topology hiding computation and specifies, for each protocol, the classes of graphs for which it is guaranteed to run in polynomial time.

The first result was a feasibility result and was the work of Moran, Orlov, and Richelson [22] from 2015. Their result was a broadcast protocol secure against static, semi-honest adversaries, and a protocol against failstop adversaries that do not disconnect the graph. However, their protocol is restricted to communication graphs with diameter logarithmic in the total number of parties.

The main idea behind their protocol is a series of nested multiparty computations, in which each node is replaced by a secure computation in its local neighborhood that simulates that node. The drawback is that in order to get full security, this virtualization needs to extend to the entire graph, but the complexity of the MPC grows exponentially with the size of the neighborhood.

Our work is also a feasibility result, but instead builds on a protocol similar to the recent Akavia-Moran paper [1], which takes a different approach. They employ ideas from cryptographic voting literature, hiding the order of nodes in the cycle by "mixing" encrypted inputs before decrypting them and adding layers of public keys to the encryption at each step. In this work, we take this layer-adding approach and apply it to random walks over all kinds of graphs instead of deterministically figuring out the path beforehand.

Other related works include a work by Hirt, et al.[18], which describes a protocol that acheives better efficiency than [22], but as it uses similar tactics, is still restricted to network graphs with logarithmic diameter. Addressing a problem different from topology-hiding, the work by Chandran et.al. [6] reduces communication complexity of secure MPC by allowing each party to communicate with a small (sublinear in the number of parties) number

Graphs families	[18, 22]	[1]	[This Work]
Log diameter constant degree	+	−	+
Cycles, trees	−	+	+
Log circumference	−	+	+
Log diameter super-constant degree	−	−	+
Regular graphs	−	−	+
Arbitrary graphs	−	−	+

Table 1.1: Comparison to previous works. Rows correspond to graph families; columns corresponds to prior works in the first two columns and to this work in last the column. A +/- mark for graph x and work y indicates that a topology hiding protocol is given/not-given in work y for graph x.

of its neighbors.

Topology Hiding in Information Theoretic Settings. Hinkelmann and Jakoby [17] considered the question of topology-hiding secure computation, but focused on the information theoretic setting. Their main result was negative: any MPC protocol in the information-theoretic setting inherently leaks information about the network graph to an adversary. However, they also show that the only information we need to leak is the routing table: if we leak the routing table beforehand, then one can construct an MPC protocol which leaks no further information.

Secure Multiparty Computation with General Interaction Patterns. One of the first works in a related setting was that of Dwork et al.[8]. In their work, they have a general network topology, and they show that given an adaptive adversary, and consider both malicious and semi-honest cases. They provide some lower bounds on how many wires these adversaries can corrupt, and their goal, instead of secure MPC, was secure communication.

Closer to our own goal, Halevi et al.[15] presented a unified framework for studying secure MPC with arbitrary restricted interaction patterns, generalizing models for MPC with specific restricted interaction patterns [14, 3, 16]. Their goal is not topology hiding, however. Instead, they ask the question of when is it possible to prevent an adversary from learning the output to a function on several inputs. They started by observing that an adversary controlling the final players $P_i, \cdots, P_n$ in the interaction pattern can learn the output of the computed function on several inputs because the adversary can rewind and execute the protocol on any possible party values $x_i, \cdots, x_n$. This model allows complete

knowledge on the underlying interaction pattern (or as in our case, the graph).

1.4 Organization of Thesis

In chapter 2 we describe our adversarial model and introduce some notation. In section 2.4 we detail the special properties we require from the encryption scheme that we use in our cycle protocol, and show how it can be instantiated based on DDH. In section 2.5, we discuss the kinds of *exploration sequences*, sequences that cover the graph, that we need for our protocol to be correct and secure: in section 2.5.1 we discuss how correlated random walks fit that description, and in section 2.5.2 we prove that universal exploration sequences also satisfy the description. In chapter 3, we define our security model, which is slightly stronger than the one in [1]. In chapter 4, we go over the compilation from topology-hiding broadcast to topology-hiding computation, using public key encryption. In chapter 5, we explain our protocol for topology-hiding broadcast on general graphs and prove its completeness and security. Then, in section 5.3, we go over a time and communication tradeoff, and explain how we can optimize our protocol with respect to certain classes of graphs. Finally, in chapter 6, we conclude and discuss future work.

Publication Information The work done in this thesis was presented at Crypto 2017 and will appear in *Advances in Cryptology - CRYPTO 2017*, pages 447-467.

Chapter 2

Preliminaries

2.1 Computation and Adversarial Models

We model a network by an undirected graph $G = (V, E)$ that is not fully connected. We consider a system with n parties denoted $P_1, \ldots, P_n$, where n is upper bounded by poly(κ) and κ is the security parameter. We identify V with the set of parties $\{P_1, \ldots, P_n\}$.

We consider a static and computationally bounded (PPT) adversary that controls some subset of parties (any number of parties). That is, at the beginning of the protocol, the adversary corrupts a subset of the parties and may instruct them to deviate from the protocol according to the corruption model. Throughout this work, we consider only semi-honest adversaries. In addition, we assume that the adversary is rushing; that is, in each round the adversary sees the messages sent by the honest parties before sending the messages of the corrupted parties for this round. For a detailed description of the general MPC definitions and descriptions of the adversarial model we use, see chapter 4, and for a more in-depth description of these models (which chapter 4 was based on), see [12].

2.2 Notation

In this section, we describe our common notation conventions for both graphs and for our protocol.

2.2.1 Graph Notation

Let $G = (V, E)$ be an undirected graph. For every $v \in V$, we define the neighbors of v as $\mathcal{N}(v) = \{w : (v, w) \in E\}$ and will refer to the degree of v as $d_v = |\mathcal{N}(v)|$.

2.2.2 Protocol Notation

Our protocol will rely on generating many public-secret key pairs, and ciphertexts at each round. In fact, each node will produce a public-secret key pair for each of its neighbors at every timestep. To keep track of all these, we introduce the following notation. Let $pk_{i\to d}^{(t)}$ represent the public key created by node i to be used for neighbor d at round t; $sk_{i\to d}^{(t)}$ is the corresponding secret key. Ciphertexts are labeled similarly: $c_{d\to i}^{(t)}$, is from neighbor d to node i.

2.3 UC Security

As in [22], we prove security in the UC model [4]. If a protocol is secure in the UC model, it can be composed with other protocols without compromising security, so we can use it as a subprotocol in other constructions. This is critical for constructing topology-hiding MPC based on broadcast—broadcast is used as a sub-protocol.

A downside of the UC model is that, against general adversaries, it requires setup. However, setup is not necessary against semi-honest adversaries that must play according to the rules of the protocol. Thus, we get a protocol that is secure in the plain model, without setup. For details about the UC framework, we refer the reader to [4].

2.4 Privately Key-Commutative and Rerandomizable Encryption

As in [1], we require a public key encryption scheme with the properties of being *homomorphic* (with respect to OR in our case), *privately key-commutative*, and *re-randomizable*.

In this section we first formally define the properties we require, and then show how they can be achieved based on the Decisional Diffie-Hellman assumption.

We call an encryption scheme satisfying the latter two properties, i.e., privately key-commutative and re-randomizable, a *PKCR-encryption*.

2.4.1 Required Properties

Let $\mathsf{KeyGen} : \{0,1\}^* \mapsto \mathcal{PK} \times \mathcal{SK}, \mathsf{Enc} : \mathcal{PK} \times \mathcal{M} \times \{0,1\}^* \mapsto C, \mathsf{Dec} : \mathcal{SK} \times C \mapsto \mathcal{M}$ be the encryption scheme's key generation, encryption and decryption functions, respectively, where $\mathcal{PK}$ is the space of public keys, $\mathcal{SK}$ the space of secret keys, $\mathcal{M}$ the space of plaintext messages and C the space of ciphertexts.

We will use the shorthand $[m]_k$ to denote an encryption of the message m under public-key k. We assume that for every secret key $sk \in \mathcal{SK}$ there is associated a single public key $pk \in \mathcal{PK}$ such that (pk, sk) are in the range of KeyGen. We slightly abuse notation and denote the public key corresponding to sk by $pk(sk)$.

Privately Key-Commutative

The set of public keys $\mathcal{PK}$ form an abelian (commutative) group. We denote the group operation $\circledast$. Given any $k_1, k_2 \in \mathcal{PK}$, there exists an efficient algorithm to compute $k_1 \circledast k_2$. We denote the inverse of k by k^{-1} (i.e., $k^{-1} \circledast k$ is the identity element of the group). Given a secret key sk, there must be an efficient algorithm to compute the inverse of its public key $(pk(sk))^{-1}$.

There exist a pair of algorithms $\mathsf{AddLayer} : C \times \mathcal{SK} \mapsto C$ and $\mathsf{DelLayer} : C \times \mathcal{SK} \mapsto C$ that satisfy:

1. For every public key $k \in \mathcal{PK}$, every message $m \in \mathcal{M}$ and every ciphertext $c = [m]_k$,

$$\mathsf{AddLayer}(c, sk) = [m]_{k \circledast pk(sk)} \ .$$

2. For every public key $k \in \mathcal{PK}$, every message $m \in \mathcal{M}$ and every ciphertext $c = [m]_k$,

$$\mathsf{DelLayer}(c, sk) = [m]_{k \circledast (pk(sk))^{-1}} .$$

We call this *privately* key-commutative since adding and deleting layers both require knowledge of the secret key.

Note that since the group $\mathcal{PK}$ is commutative, adding and deleting layers can be done in any order.

Rerandomizable

We require that there exists a ciphertexts "re-randomizing" algorithm $\mathsf{Rand} : C \times \mathcal{PK} \times \{0,1\}^* \mapsto C$ satisfying the following:

1. *Randomization:* For every message $m \in \mathcal{M}$, every public key $pk \in \mathcal{PK}$ and ciphertext $c = [m]_{pk}$, the distributions $(m, pk, c, \mathsf{Rand}(c, pk, U^*))$ and $(m, pk, c, \mathsf{Enc}_{pk}(m; U^*))$ are computationally indistinguishable.

2. *Neutrality:* For every ciphertext $c \in C$, every secret key $sk \in \mathcal{SK}$ and every $r \in \{0,1\}^*$,

$$\mathsf{Dec}_{sk}(c) = \mathsf{Dec}_{sk}(\mathsf{Rand}(c, pk(sk), r)) .$$

Furthermore, we require that public-keys are "re-randomizable" in the sense that the product $k \circledast k'$ of an arbitrary public key k with a public-key k' generated using KeyGen is computationally indistinguishable from a fresh public-key generated by KeyGen.

Homomorphism

We require that the message space $\mathcal{M}$ forms a group with operation denoted $\cdot$, and require that the encryption scheme is homomorphic with respect this operation $\cdot$ in the sense that there exists an efficient algorithm $\mathsf{hMult} : C \times C \mapsto C$ that, given two ciphertexts $c = [m]_{pk}$ and $c' = [m']_{pk}$, returns a ciphertext $c'' \leftarrow \mathsf{hMult}(c_1, c_2)$ s.t. $\mathsf{Dec}_{sk}(c'') = m \cdot m'$ (for sk the secret-key associated with public-key pk).

Notice that with this operation, we can homomorphically raise any ciphertext to any power via repeated squaring. We will call this operation hPower.

Homomorphic OR

This feature is built up from the re-randomizing and the homomorphism features. One of the necessary parts of our protocol for broadcast functionality is to have a homomorphic OR. We need this operation not to reveal if it is ORing two 1's or one 1 at decryption. So, following [1], first we define an encryption of 0 to be an encryption of the identity element in $\mathcal{M}$ and an encryption of 1 to be an encryption of any other element. Then, we define HomOR so that it re-randomizes encryptions of 0 and 1 by raising ciphertexts to a random power with hPower.

---

function HomOR($c, c', pk, r = (r, r')$) // r is randomness
  $\hat{c} \leftarrow$ hPower (c, r, pk) and $\hat{c}' \leftarrow$ hPower (c', r', pk)
  return Rand (hMult $(\hat{c}, \hat{c}'')$, pk)
end function

Claim 2.4.1. *Let $\mathcal{M}$ have prime order p, where $1/p$ is negligible in the security parameter, and $M, M' \in \{0, 1\}$ be messages with corresponding ciphertexts c and c' under public key pk. The distribution* $(c, c', pk, M, M', \mathsf{Enc}(M \vee M', pk; U^*))$ *is computationally indistinguishable from*

$(c, c', pk, M, M', \mathsf{HomOR}(c, c', pk; U^*))$.

Proof. We will go through three cases for values of M and M': first, when $M = M' = 0$; second when $M = 1$ and $M' = 0$; and third when $M = 1$ and $M' = 1$. The case $M = 0$ and $M' = 1$ is handled by the second case.

- Consider when $M = M' = 0$. Note that $1_{\mathcal{M}}$ is the group element in $\mathcal{M}$ that encodes 0, so an encryption of 0 is represented by an encryption of the identity element, $m = m' = 1_{\mathcal{M}}$, of $\mathcal{M}$. Consider c_0 and c_0' both encryptions of $1_{\mathcal{M}}$. After hPower, both $\hat{c}_0$ and $\hat{c}_0'$ are still encryptions of $1_{\mathcal{M}}$. hMult then produces an encryption of $1_{\mathcal{M}} \cdot 1_{\mathcal{M}} = 1_{\mathcal{M}}$, and Rand makes that ciphertext indistinguishable to a fresh encryption of $1_{\mathcal{M}}$. We have proved our first case.

- Next, let c_0 be an encryption of 0 and c'_1 be an encryption of 1. In this case, 0 is represented again by $1_{\mathcal{M}}$, but c'_1 is represented by some $m' \xleftarrow{\$} \mathcal{M}$ (with all but negligible probability $m' \neq 1$). After hPower, $\hat{c}_0$ still encrypts $1_{\mathcal{M}}$, but $\hat{c}'_1$ encrypts $\hat{m} = m'^{r'}$ for some $r' \xleftarrow{\$} \mathbb{Z}_p$. hMult yeilds an encryption of $\hat{m}$ and Rand makes a ciphertext computationally indistinguishable from a fresh encryption of $\hat{m}$. Since $\mathcal{M}$ has prime order p and $r' \xleftarrow{\$} \mathbb{Z}_p$, as long as $m' \neq 1$, m'^{r} is uniformly distributed over $\mathcal{M}$, and so computationally has a distribution indistinguishable to a fresh encryption of the boolean message 1.

- Finally, let c_1 and c'_1 both be encryptions of 1: c_1 encrypts $m \xleftarrow{\$} \mathcal{M}$ and c'_1 encrypts $m' \xleftarrow{\$} \mathcal{M}$. We will go through the same steps to have at the end, a ciphertext computationally indistinguishable[1] from a fresh encryption of $m^r \cdot m'^{r'}$ for $r, r' \xleftarrow{\$} \mathbb{Z}_p$. Again because the order of $\mathcal{M}$ is prime, $m^r \cdot m'^{r'}$ is uniformly distributed over $\mathbb{Z}_p$, and so the resulting ciphertext looks like a fresh encryption of 1.

□

This claim means that we cannot tell how many times 1 or 0 has been OR'd together during an or-and-forward type of protocol. This will be critical in our proof of security.

Instantiation of OR-homomorphic PKCR-enc under DDH

We use standard ElGamal, augmented by the additional required functions. The KeyGen, Dec and Enc functions are the standard ElGamal functions, except that to obtain a one-to-one mapping between public keys and secret keys, we fix the group G and the generator g, and different public keys vary only in the element $h = g^x$. Below, g is always the group generator. The Rand function is also the standard rerandomization function for ElGamal:

function RAND($c = (c_1, c_2), pk, r$)

return $(c_1 \cdot g^r, pk^r \cdot c_2)$

end function

[1] In our definition of a PKCR encryption scheme, Rand is only required to be computationally randomizing, which carries over in our distribution of homomorphically-OR'd ciphertexts. However, ElGamal's re-randomization function is distributed statistically close to a fresh ciphertext, and so our construction will end up having HomOR be identically distributed to a fresh encryption of the OR of the bits.

We use the shorthand notation of writing Rand (c, pk) when the random coins r are chosen independently at random during the execution of Rand. We note that the distribution of public-keys outputted by KeyGen is uniform, and thus the requirement for "public-key rerandomization" indeed holds. ElGamal public keys are already defined over an abelian group, and the operation is efficient. For adding and removing layers, we define:

function AddLayer$(c = (c_1, c_2), sk)$

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

end function

function DelLayer$(c = (c_1, c_2), sk)$

  return $(c_1, c_2 / c_1^{sk})$

end function

Every ciphertext $[m]_{pk}$ has the form $(g^r, pk^r \cdot m)$ for some element $r \in \mathbb{Z}_{ord(g)}$. So

$$\mathsf{AddLayer}\left([m]_{pk}, sk'\right) = (g^r, pk^r \cdot m \cdot g^{r \cdot sk'}) = (g^r, pk^r \cdot (pk')^r \cdot m) = (g^r, (pk \cdot pk')^r \cdot m) = [m]_{pk \cdot pk'} .$$

It is easy to verify that the corresponding requirement is satisfied for DelLayer as well.

ElGamal message space already defined over an abelian group with homomorphic multiplication, specifically:

function hMult$(c = (c_1, c_2), c' = (c'_1, c'_2))$

  return $c'' = (c_1 \cdot c'_1, c_2 \cdot c'_2)$

end function

Recalling that the input ciphertext have the form $c = (g^r, pk^r \cdot m)$ and $c' = (g^{r'}, pk^{r'} \cdot m')$ for messages $m, m' \in \mathbb{Z}_{ord(g)}$, it is easy to verify that decrypting the ciphertext $c'' = (g^{r+r'}, pk^{r+r'} \cdot m \cdot m')$ returned from hMult yields the product message $\mathsf{Dec}_{sk}(c'') = m \cdot m'$.

Finally, to obtain a negligible error probability in our broadcast protocols, we take G a prime order group of size satisfying that $1/|G|$ is negligible in the security parameter κ. With this property and valid Rand and hMult operations, we get hPower and hence HomOR with ElGamal.

2.5 Graph Exploration Sequences

A key element in designing our algorithm is an *exploration sequence*. Informally, it is a sequence of edges that, when given an edge to start on, traverses the entire graph. Consider the following way to define a walk on a d-regular graph. Given a sequence $\tau_1, \ldots, \tau_T \in \{0, \ldots, d-1\}$ and starting edge $e_0 = (v_{-1}, v_0)$, we define the walk $v_1, \ldots, v_T$ as follows: if we enter node v_i from edge s, we leave v_i to v_{i+1} on edge $s+\tau_i \mod d$. If the walk $v_1, \ldots, v_T$ covers the entire graph (regardless of the starting node), then it is an exploration sequence. In this section, we will formally define these objects and then provide two methods for constructing them.

Definition 2.5.1. An *exploration sequence* for d-regular graphs on n nodes is a sequence $\tau_1, \ldots, \tau_T \in \{0, \ldots, d-1\}$ and starting edge $e_0 = (v_{-1}, v_0)$ so that the resulting walk $v_1, \ldots, v_T$ covers all d-regular graphs on n nodes.

We can also define an exploration sequence with error and for non-regular graphs, just based on the maximum degree d, or n for any graph. In this case, our walk is still just defined by a starting edge and offsets $\tau_1, \ldots, \tau_T \in \{0, \ldots, d\}$; if the τ_i's are generated randomly, then there could be some probability that the walk fails. Now, if we enter node v_i from edge s, and node v_i has degree d_i, we take edge $s + \tau_i \mod d_i$ to node v_{i+1}.

Definition 2.5.2. An *δ-exploration sequence* for n-node graphs with maximum degree d is a sequence $\tau_1, \ldots, \tau_T \in \{0, \ldots, d-1\}$ and starting edge $e_0 = (v_{-1}, v_0)$ so that the resulting walk $v_1, \ldots, v_T$ covers any n-node graph with max degree d with probability at least $1 - \delta$ over the randomness used to generate the sequence.[2]

To get correctness, we will need to run at least one of these walks per node. For topology-hiding, we need to have one walk per direction on each edge (so a total of 2·|edges| walks). Moreover, at every step in this collection of sequences, we want exactly one of these walks to be going down a direction of an edge: we do not want the walks to "interfere" with each other. For instance, imagine that each undirected edge is actually two pipes from each

[2]Note that the probability that the sequence covers the graph is based on the randomness used to define the sequence. Some exploration sequences may not be randomly generated; and then they would either cover all such graphs or have error probability 1.

of the nodes: one pipe going from the first to the second and the other pipe from the second to the first and each pipe has the capacity for one walk at each round; there can only be one walk occupying one direction at each step.

Definition 2.5.3. A *non-interfering* collection of exploration sequences for graphs on n nodes with m edges is group of $2m$ exploration sequences for graphs on n nodes such that if they run simultaneously the do not interfere: no two sequences ever walk the same direction down the same edge.

So, a non-interfering full collection of covering sequences has one sequence traversing down each direction of each edge at every step. Because of this non-interference property, at each node, we can model each step of the sequences as a permutation on that node's edges. That is, for every node v in every round $t \in [T]$, there exists a permutation $\pi_{v,t}$ on that node's edges. These permutations describe every walk: if a walk enters a node from edge i at round t, it leaves that node from edge $\pi_{v,t}(i)$. So, we define the following function:

$$\mathsf{Seq} : V \times [T] \times [d] \to S_{d_v}$$
$$\mathsf{Seq} : (v, t, d_v) \mapsto \pi_{v,t}$$

Because our resulting protocol is topology hiding, a node's local view cannot rely on the topology of the graph to generate its permutation. The function Seq needs to be generated *information-locally*. That is, a node needs to be able to compute $\mathsf{Seq}(v, t, d_v)$ using only its own local information it has on itself and its neighbors; Seq is an *information-local function.*

Definition 2.5.4 ([1])**.** A function computed over a graph $G = (V, E)$ is *information-local* if the output of every node $v \in V$ can be computed from its own input and random coins.[3]

Altogether, we will need a full collection of exploration sequences that is non-interfering and information-local. Correlated random walks, for example, fit this description. We will also show that a deterministic, polynomial-time constructible object (universal exploration

[3]The definition proposed by [1] generalizes this one with k-information-local functions. We only care about 0-information-local functions for this work.

sequences) also fit this description. We will analyze and compare how efficient these objects are in section 5.3 (i.e. how many steps each of them takes).

2.5.1 Correlated Random Walks

Here we will prove that correlated random walks of length $O(\kappa \cdot n^3)$ are an example of a collection of non-interfering $\text{negl}(\kappa)$-exploration sequences, assuming $n = \text{poly}(\kappa)$. By the main theorem, theorem 5.0.3, this will imply an instantiation of our protocol that uses random walks. In order to prove this, we will first need to prove some qualities about the random walks. We will rely on the following definition and theorem from Mitzenmacher and Upfal's book (see chapter 5)[21].

Definition 2.5.5 (Cover time)**.** The cover time of a graph $G = (V, E)$ is the maximum over all vertices $v \in V$ of the expected time to visit all of the nodes in the graph by a random walk starting from v.

Theorem 2.5.6 (Cover time bound)**.** *The cover time of any connected, undirected graph $G = (u, v)$ is bounded above by $4nm \leq 4n^3$.*

Corollary 2.5.7. *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 $\tau \cdot (8n^3)$ steps. We have*

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

Proof. First, consider a random walk that takes t steps to traverse a graph. Theorem 2.5.6 tells us that we expect $t \leq 4n^3$, and so by a Markov bound, we have

$$\Pr\left[t \geq 2 \cdot (4n^3)\right] \leq \frac{1}{2}$$

Translating this into our notation, for any node $u \in G$, $\Pr[\mathcal{W}(u, 1) = V] \geq \frac{1}{2}$.

We can represent $\mathcal{W}(u, \tau)$ as a union of τ random walks, each of length $8n^3$: $\mathcal{W}(u_1 = u, 1) \cup \mathcal{W}(u_2, 1) \cup \cdots \cup \mathcal{W}(u_\tau, 1)$, where u_i is the node we have reached at step $i \cdot 8n^3$ (technically, u_i is a random variable, but the specific node at which we start each walk will

not matter). $\mathcal{W}(u, \tau)$ will succeed in covering all nodes in G if any $\mathcal{W}(u_i, 1)$ covers all nodes.

So, we will bound the probability that all $\mathcal{W}(u_i, 1) \neq V$. Note that each $\mathcal{W}(u_i, 1)$ is independent of all other walks except for the node it starts on, but our upper bound is independent of the starting node. This means

$$\Pr[\mathcal{W}(u_i, 1) \neq V, \forall i \in [\tau]] = \prod_{i \in [\tau]} \Pr[\mathcal{W}(u_i, 1) \neq V] \leq \frac{1}{2^\tau}.$$

Therefore,

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

□

Lemma 2.5.8. *A full collection of correlated random walks of length $\kappa \cdot 8n^3$ is a full collection of non-interfering $2^{-\kappa}$-exploration sequences.*

Proof. We already know that correlated random walks are non-interfering by definition. By corollary 2.5.7, we also know that each walk has probability $2^{-\kappa} = \text{negl}(\kappa)$ of covering the entire graph. The lemma follows immediately. □

2.5.2 Perfect Covering: Universal Exploration Sequences

In this section we will prove that Universal Exploration Sequences (UESs) are also a full collection of non-interfering covering sequences. Unlike random walks, however, these are deterministic walks that are guaranteed to cover the entire graph. We will see in section 5.3.2 that while these exploration sequences are guaranteed to hit every node in the graph, we do not have good bounds on the length of polynomial-time computable exploration sequences (only that we can compute them in polynomial time, and they will be polynomial in length).

UESs are typically just described for d-regular graphs, but that is mostly because any general graph can be transformed into a 3-regular graph using a transformation by Koucky [20].

Definition 2.5.9. A *universal exploration sequence* for d-regular graphs of size n is a sequence of instructions $\tau_1, \ldots, \tau_T \in \{0, 1, \ldots, d-1\}$ so that if for every connected d-regular graph $G = (V, E)$ on n vertices, any number of its edges, and any starting edge $(v_{-1}, v_0) \in E$, then walk visits all vertices in the graph.

Work by both Koucky and Reingold show that exploration sequences for any graph exist, are polynomial in length, and can be computed in polynomial time.

Lemma 2.5.10 ([25]). *There exists a polynomial length exploration sequence for any graph G on n nodes which can be computed in polynomial time.*

So, what we have is a sequence that every node in a graph can compute locally and in polynomial time. We just need to prove that if we run these UESs simultaneously, they will not interfere.

Lemma 2.5.11. *A full collection of identical universal exploration sequences (UESs) for graphs on n nodes is a full collection of non-interfering information-local* 0*-exploration sequences.*

Proof. By definition, we know that every one of the exploration sequences in the collection will explore the entire graph, so they are 0-exploration sequences (have 0 chance of error). Also note that from lemma 2.5.10, each party can locally compute the identical sequence in polynomial time.

We only need to prove that these walks will not interfere with each other. We know that at the first step of the algorithm, no sequences will interfere since they all start at different edges or directions down an edge. So, consider that no walks have interfered at step t, and consider node i with degree d_i. Node i has d_i walks entering at time t. Each walk has the same relative instruction at time t: τ_t. So, a walk entering from edge e will leave edge $e + \tau_t \mod d_i$. For two walks to collide, $e + \tau_t = e' + \tau_t \mod d_i$, implying $e = e' \mod d_i$. Since $0 \leq e, e' < d_i$, we get that $e = e'$, contradicting that no walks were interfering before this step. Therefore, none of these walks will interfere. □

Chapter 3

A Stronger Simulation-based Definition of Topology-Hiding

Here we adapt the simulation-based definition of topology-hiding from [22] to be even stronger: the simulator only needs to know pseudonyms for each neighbor of a party, instead of exactly which parties correspond to which neighbors (in [22]). This definition will also be in the UC framework, and our discussion of it will be much the same.

The UC model usually assumes all parties can communicate directly with all other parties. To model the restricted communication setting, [22] define the $\mathcal{F}_{\text{graph}}$-hybrid model, which employs a special "graph party," P_{graph}. Figure 3.0.1 shows $\mathcal{F}_{\text{graph}}$'s functionality: at the start of the functionality, $\mathcal{F}_{\text{graph}}$ receives the network graph from P_{graph}, and then outputs, to each party, that party's neighbors. Then, $\mathcal{F}_{\text{graph}}$ acts as an "ideal channel" for parties to communicate with their neighbors, restricting communications to those allowed by the graph.

Since the graph structure is an input to one of the parties in the computation, the standard security guarantees of the UC model ensure that the graph structure remains hidden (since the only information revealed about parties' inputs is what can be computed from the output). Note that the P_{graph} party serves only to specify the communication graph, and does not otherwise participate in the protocol.

In our definition, $\mathcal{F}_{\text{Graph}}$ recieves the graph from P_{graph} (as in [22]), but –unlike [22] – $\mathcal{F}_{\text{Graph}}$ does not output the neighbors to each party. Instead, $\mathcal{F}_{\text{graph}}$ reveals edge-labels. These

Participants/Notation:

This functionality involves all the parties $P_1, \ldots, P_n$ and a special graph party P_{graph}.

Initialization Phase:

Inputs: $\mathcal{F}_{\text{graph}}$ waits to receive the graph $G = (V, E)$ from P_{graph}, and $\mathcal{F}_{\text{graph}}$ constructs a random injective function $f : E \to [n^2]$, labeling each edge with an element from $[n^2]$.

Outputs: For each node v, $\mathcal{F}_{\text{graph}}$ gives the set of edge labels $L_v = \{f(u, v) : (u, v) \in E\}$ to P_v.

Communication Phase:

Inputs: $\mathcal{F}_{\text{graph}}$ receives from a party P_v a destination/data pair (ℓ, m) where $f(v, w) = \ell \in L_v$ indicates to $\mathcal{F}_{\text{graph}}$ neighbor w, and m is the message P_v wants to send to P_v.

Output: $\mathcal{F}_{\text{graph}}$ gives output (ℓ, m) to P_w, where $f(v, w) = \ell$, indicating that the neighbor on edge ℓ sent the message m to P_w.

Figure 3.0.1: The functionality $\mathcal{F}_{\text{graph}}$ with edge labels. Note that since the graph is undirected, $(u, v) = (v, u) \in E$ and so $f(u, v) = f(v, u)$.

labels act as pseudonyms when one node wants to communicate with another, but without revealing which party corresponds to which neighbor. So, we leak enough information for nodes to tell if they share an edge wtih another node, but not enough to be able to tell if two nodes share a neighbor. We capture this leak information to any ideal-world adversary in the functionality $\mathcal{F}_{\text{graphInfo}}$, which is just the initialization phase of $\mathcal{F}_{\text{graph}}$. For any other functionality $\mathcal{F}$ we want to model in the ideal world, we compose $\mathcal{F}$ with $\mathcal{F}_{\text{graphInfo}}$, writing $(\mathcal{F}_{\text{graphInfo}}||\mathcal{F})$.

Now we can define topology-hiding MPC in the UC framework:

Definition 3.0.12. We say that a protocol Π is a topology-hiding realization of a functionality $\mathcal{F}$ if it UC-realizes $(\mathcal{F}_{\text{graphInfo}}||\mathcal{F})$ in the $\mathcal{F}_{\text{graph}}$-hybrid model.

Our definition also captures functionalities that depend on the structure of the graph, like shortest path or determining the length of the longest cycle.

3.1 Differences of this Model: Neighbors of Neighbors

In the first model, proposed by [22], $\mathcal{F}_{\text{graphInfo}}$ reveals exactly the neighbors of each party P_v. This means that if an adversary controls two nodes, he can tell if they have a common neighbor. In this model, we reveal edge labels instead of the explicit edges, and since the label is only shared between those two nodes that have that edge, corrupted nodes cannot tell if they have a common neighbor, unless that neighbor is also corrupted.

3.2 Broadcast functionality, $\mathcal{F}_{\text{Broadcast}}$

In accordance with this definition, we need to define an ideal functionality of broadcast, denoted $\mathcal{F}_{\text{Broadcast}}$, shown in figure 3.2.1. We will prove that a simulator only with knowledge of the output of $\mathcal{F}_{\text{Broadcast}}$ and knowledge of the local topology of the adversarially chosen nodes Q can produce a transcript to nodes in Q indistinguishable from running our protocol.

Participants/Notation:
This functionality involves all the parties $P_1, \ldots, P_n$.

Inputs: The broadcasting party P_i receives a bit $b \in \{0, 1\}$.

Outputs: All parties $P_1, \ldots, P_n$ receive output b.

Figure 3.2.1: The functionality $\mathcal{F}_{\text{Broadcast}}$.

Chapter 4

From Broadcast to Secure Multiparty Computation

Our method for proving that topology-hiding computation (THC) is possible involves compiling general MPC protocols using UC-secure topology-hiding broadcast (THB) and public-key cryptography. In this chapter, we will go into detail about the model of MPC we realize (semi-honest adversaries statically corrupting any subset of nodes with synchronous communication). Then, we will formally prove that UC-secure THB along with public key cryptography can compile any MPC protocol in this model into a topology-hiding MPC using our security definition, detailed in chapter 3.

4.1 The MPC and THB models

In this section, we go over our exact security models of what we need to achieve THC. First, we will describe the standard MPC model which is synchronous and secure against semi-honest adversaries. Then, we will adapt our definition for what UC-secure THB is, mainly so that it works well with the proof that our compilation of THB to THC works. Finally, we note that we need CPA-secure public key encryption (secure against only chosen plaintext attacks), and provide a definition for it.

4.1.1 MPC model: Semi-honest adversaries

The material in this section is referenced one of the MPC models described by Goldreich in [12].

First we will explain some of our notation. The goal will be to compute an n-ary function $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, where each of the n inputs correspond to an input from one of the parties taking part in the computation, and the outputs will correspond to the output a party receives.

Let Π denote some protocol for n parties. Π assumes synchronous communication and point-to-point channels between each party. Every round, parties send and receive messages from each other along these channels, and then perform some computations on them. For a function f with inputs $\mathbf{x} = (x_1, \ldots, x_n)$ respectively from parties $P_1, \ldots, P_n$, Π *realizes* the functionality of f if by the end of the protocol, each party i gets the output $f(\mathbf{x})_i$.

Definition 4.1.1. For a protocol Π for n parties, the view of a party is

$$\text{VIEW}_i^{\Pi}(\mathbf{x}) = (x_i, r, m_1, m_T)$$

and the view of any subsets of parties $I \subset [n]$ is

$$\text{VIEW}_I^{\Pi}(\mathbf{x}) = (I, (\text{VIEW}_i^{\Pi}(\mathbf{x}))_{i \in I}).$$

The outputs our defined similarly:

$$\text{OUTPUT}_i^{\Pi}(\mathbf{x}) = f_i(x), \text{ and } \text{OUTPUT}_I^{\Pi}(\mathbf{x}) = (f_i(x))_{i \in I}$$

Definition 4.1.2. For a protocol Π realizing a functionality f, we say Π *privately computes* f if there exists a PPT algorithm $\mathcal{S}$ (a simulator), such that for all subsets $I \subset [n]$,

$$\{\mathcal{S}(I, (x_i)_{i \in I}, f_I(\mathbf{x})), f(\mathbf{x})\} \overset{c}{\approx} \{\text{VIEW}_I^{\Pi}(\mathbf{x}), \text{OUTPUT}^{\Pi}\}$$

This notion of being *private computatable* states exactly that if a PPT adversary cor-

rupting some subset I of the parties, but follows the protocol Π (is semihonest), then she has a negligible chance of distinguishing between the world where she is interacting with other parties and in the world where she interacts with the simulator $\mathcal{S}$. This is equivalent to our notion of secure MPC throughout this work.

4.1.2 THC Model

Here we will review what it takes for a protocol to be topology hiding. The formal definition, from definition 3.0.12, states that we need a protocol that UC-realizes $(\mathcal{F}_{\text{graphInfo}} \| \mathcal{F})$ in the $\mathcal{F}_{\text{graph}}$-hybrid model. Let $\mathcal{F}_{\text{graphInfo}}(I)$ represent the local graph information of parties in I in accordance with the functionality of $\mathcal{F}_{\text{graph}}$. So, we say that for a protocol to be a topology-hiding realization of a function f, there exists a PPT simulator $\mathcal{S}$ that only has access to the local graph information and local computation information to produce views computationally indistinguishable from views in the real protocol. That is, against a static, semi-honest adversary, we just need the following distributions to be computationally indistinguishable in order for a protocol to be topology hiding: for any subset of parties $I \subset [n]$,

$$\{\mathcal{S}(I, \mathcal{F}_{\text{graphInfo}}(I), (x_i)_{i\in I}, f_i(\mathbf{x})), f(\mathbf{x})\}_{\mathbf{x}\in(\{0,1\}^*)^n} \overset{c}{\approx} \{\text{VIEW}_I^{\Pi}(\mathbf{x}), \text{OUTPUT}^{\Pi}(\mathbf{x})\}_{\mathbf{x}\in(\{0,1\}^*)^n}$$

For an in-depth description of the UC-model and for why this definition is UC, we refer the reader to Canettti's work on universal-composability [4].

4.1.3 CPA-Secure Public Key Encryption

We will need one more element to go from THB to THC: a public key encryption scheme secure against plaintext attacks (CPA-secure PKE). For completeness, we have included a definition here.

Definition 4.1.3. A public key encryption scheme (KeyGen, Enc, Dec) is CPA secure if any PPT adversary $\mathcal{A}$ cannot win the IND-CPA security game with probability greater than $1/2 + \text{negl}(\kappa)$.

Now we define this security game:

Definition 4.1.4. The IND-CPA security game works as follows:

1. Setup. The challenger C gets public and secret keys $(pk, sk) \leftarrow \mathsf{KeyGen}(\kappa)$ and sends pk to $\mathcal{A}$.

2. Challenge phase. The adversary performs as many encryptions as she wants using pk and then sends challenge messages M_0 and M_1. C chooses a random bit b and sends the ciphertext $\mathsf{Enc}(pk, M_b)$ to $\mathcal{A}$.

3. Output phase. $\mathcal{A}$ outputs b' and wins if $b = b'$.

4.2 Compiling MPC to Topology hiding MPC with Broadcast and Public Key Encryption

In this section we will prove that with THB and CPA-secure PKE, we get a topology-hiding realization of any MPC protocol Π. Since there exist MPC protocols against static, semi-honest adversaries for all efficiently computable functions, it follows that we get topology-hiding computation for all efficiently computable functions.

Theorem 4.2.1 (Compiling THC from THB and PKE). *Assume UC-secure THB and CPA-secure PKE exist. Then, for any PPT protocol* Π *that privately computes a function* $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, *there exists a PPT protocol* Π' *that is a topology-hiding realization of the functionality of* f.

Proof. Π is a PPT multiparty protocol: instructions are either to run a local computation or, at each round, to send some messages from one party to another along a point-to-point channel. Π' will operate as Π except there will be a setup phase and point-to-point communication will be handled with broadcast and public-key encryption.

Let ϕ be the topology-hiding broadcast protocol. Π' works as follows:

- Setup phase. Every party creates a public-secret key pair (pk_i, sk_i). Then, every party broadcasts their public key pk_i via ϕ.

- Point-to-point communication. If party i needs to send a message m to party j in protocol Π, Π' dictates the following. First, party i computes $c_i \leftarrow \mathsf{Enc}(pk_j, m)$. Then, party i broadcasts c_i using ϕ under a session ID corresponding to that channel. Finally, party j, upon receiving c_i, decrypts $m_i \leftarrow \mathsf{Dec}(sk_i, c_i)$.

- Internal computation. Any internal computation run on information gathered from other parties or from randomness is carried out exactly as in Π.

Now we have to prove that Π' realizes the ideal, topology-hiding functionality of Π. First, Π' is correct. This follows from correctness of Π in computing the functionality and from the correctness of encryption and decryption.

Proving that this is topology-hiding is more involved. Since Π privately computes f, there exists a simulator $\mathcal{S}$ so that for any adversary controlling $I \subset [n]$ parties, $\mathcal{S}$ can simulate the views of the adversary without any knowledge of the other parties' inputs. We will show that there exists a simulator $\mathcal{S}'$ simulating an adversary's view of Π' and furthermore that $\mathcal{S}'$ requires no knowledge of other parties' inputs or the structure of the graph beyond the adversary's local neighborhood. This will prove that Π' is a topology-hiding MPC.

First, let's examine the view of any subset I of parties, comparing the view of Π and the view of Π':

$$\text{VIEW}_I^{\Pi}(\mathbf{x}) = ((x_i)_{i\in I}, r, m_1, \ldots, m_T)$$
$$\text{VIEW}_I^{\Pi'}(\mathbf{x}) = ((x)_{i\in I}, r, r', R_1, \ldots R_T)$$

where each R_i is actually a collection of messages representing the communication at round i in the original protocol Π. So, we can split R_i into the communication for each point-to-point channel using the session ID's. We will show, with a series of hybrids, that we can create a simulated view computationally indistinguishable from the actual view of Π'.

- Hybrid 0. The simulator $\mathcal{S}'$ emulates the real-world view exactly. $\mathcal{S}$ requires all party inputs and the structure of the graph G. Here we will write $\mathcal{S}'_0$ as the simulator that

emulates the real-world view exactly, so

$$\{(\mathcal{S}'_0(I, \mathcal{F}_{\text{graphInfo}}([n]), \mathbf{x}, f(\mathbf{x}), f(\mathbf{x})\} \equiv \{\text{VIEW}_I^{\Pi}(\mathbf{x}), \text{OUTPUT}_I^{\Pi}(\mathbf{x})\}$$

- Hybrid 1.1 to 1.n. In these hybrids, we examine the setup phase and, instead of having our simulator use G to compute the topology hiding broadcast for each key pk_i for party $i \in [n]$, we replace it with a simulated broadcast from $\mathcal{S}_{THB}(I, \mathcal{F}_{\text{graphInfo}}(I), (x_i)_{i\in I}, f(\mathbf{x}) = pk_j)$, which does not require knowing the graph structure beyond $\mathcal{F}_{\text{graphInfo}}(I)$. Formally, the simulator in hybrid 1.j is identical to the simulator in hybrid 1.$(j-1)$ except that it simulates communication in the topology-hiding broadcast for broadcasting key pk_j with $\mathcal{S}_{THB}$.

- Hybrids 2.1 to 2.n. We still need to account for the keys broadcast during the setup phase: $\mathcal{S}'$ still needs to know what public and secret keys each each party has. $\mathcal{S}'$ now replaces the public keys generated by other parties with public keys that $\mathcal{S}'$ generates with KeyGen and ignores the secret keys of all parties $j \notin [I]$. More explicitly, for each $j \in [n]$, $j \notin I$, $\mathcal{S}'$ replaces the input pk_j to the setup phase key broadcast with a key from KeyGen. Each hybrid in this part corresponds to $j \in [n]$ (notice if $j \in I$, hybrid 2.$(j-1)$ is equivalent to hybrid 2.j).

  Notice now that for the setup phase of our compiled algorithm, $\mathcal{S}'$ does not need any information outside of $\mathcal{F}_{\text{graphInfo}}(I)$ and the inputs from parties in I.

- Hybrids 3.1 to 3.n^2. In these hybrids, $\mathcal{S}'$ replaces the real-world communication dictated by the topology-hiding broadcast protocol ϕ with simulated messages using the topology-hiding broadcast simulator $\mathcal{S}_{THB}$. That is, for each of the n^2 possible channels representing communication between i and j, we replace the perfectly simulated broadcast with messages from $\mathcal{S}_{THB}(I, \mathcal{F}_{\text{graphInfo}}(I), (x_i)_{i\in I}, f_{i\to j}(\mathbf{x}) = m_{i\to j})$.

  So, $\mathcal{S}'$ no longer requires knowing any of the topological information of the graph (all information was communicated via broadcast, and now all broadcasts have been replaced with messages from a simulator that does not need extra topological information). So, $\mathcal{S}'$ takes as input $I, \mathcal{F}_{\text{graphInfo}}(I), \mathbf{x}$, and $f(\mathbf{x})$ (notice that $\mathcal{S}'$ still depends

on inputs from parties not in I).

- Hybrids 4.1 to 4.T. Notice that our simulator in hybrid 3.n^2 still requires knowing each of the messages that every party sends to every other party, so that it can give the correct input to the broadcast subroutines. For each $t \in [T]$, hybrid 4.t will look exactly like 4.$(t-1)$ except if m_t is encrypted under pk_j for some $j \notin I$, we replace it with an encryption of 0.

- Hybrid 5. In hybrid 4.T, we require knowing all parties' messages so that we can compute the correct messages for parties in I. In this hybrid, we change all messages received by parties in I to simulated messages using the simulator $\mathcal{S}$ for the multi-party protocol Π, which takes as input $\mathcal{S}(I, (x_i)_{i\in I}, (f_i(\mathbf{x}))_{i\in I}))$. This completes the task of eliminating the simulator's need to see real messages or inputs from parties *not* in I. Now, the simulator only needs to take as input $\mathcal{S}(I, \mathcal{F}_{\text{graphInfo}}(I), (x_i)_{i\in I}, (f_i(\mathbf{x}))_{i\in I})$.

So, by the end of these hybrids, our simulator only needs local information about the corrupted parties I. Now we will prove that each hybrid is indistinguishable from its neighboring hybrids, which will finish the proof that Π' is topology-hiding.

- Hybrid 0 is computationally indistinguishable from hybrid 1. For any subset of parties I, $\mathcal{S}_{THB}$ produces a set of messages simulating the broadcast of pk_1. The set of simulated messages will be computationally indistinguishable from the parties of I interacting with the real world from the definition of topology-hiding.

- Hybrid 1.i is computationally indistinguishable from hybrid 1.$(i+1)$. This is for the same reason as before. Replacing the broadcast of public key pk_{i+1} with simulated messages from $\mathcal{S}_{THB}$ is computationally indistinguishable from the real-world communication for any subset of parties I.

- Hybrid 1.n is indistinguishable from hybrid 2.1. We're just replacing a public key generated by a party's own randomness with the simulator's randomness. The distribution of public keys will be identical.

- Hybrid 2.i is indistinguishable from hybrid 2.$(i+1)$. This is true for the same reason as above.

- Hybrid 2.n is computationally indistinguishable from hybrid 3.1. Here we replace one message channel with simulated messages from $\mathcal{S}_{THB}$. Since the message channel was represented by a broadcast, we get the same functionality, and the output from simulator $\mathcal{S}_{THB}$ will be computationally indistinguishable from the real-world views by the definition of topology hiding. It is important to note that we are running many of these broadcasts, one after another, but since we have a broadcast secure in the UC model against a passive adversary, running multiple of them simultaneously keeps the topology-hiding and privacy properties.

- Hybrid 3.i is indistinguishable from hybrid 3.$(i+1)$. This is true for the same reason as above: changing a channel from real-world communication to the simulated communication from $\mathcal{S}_{THB}$ is computationally indistinguishable to any PPT adversary controlling parties in I.

- Hybrid 3.n^2 is computationally indistinguishable from hybrid 4.1. Here we may replace an encryption of an actual message with an encryption of 0. If m_1 in the original protocol Π is sent to $j \in I$, then there is no change between the hybrids, so they will be indistinguishable to an adversary controlling parties in I. However, if m_1 is sent to $j \notin I$, then c_1 becomes an encryption of 0. The broadcast means that VIEW_I includes c_1. However, because no parties in I have a secret key associated with c_1, even an adversary controlling all parties in I could not distinguish between the two hybrids without breaking the IND-CPA security of the encryption.

- Hybrid 4.i is computationally indistinguishable from hybrid 4.$(i+1)$. This is true for the same reason as above. Either m_{i+1} is sent to a party in I, so there is no change between these hybrids, or m_{i+1} is sent to a party not in I, and the IND-CPA security of the encryption allows us to get away with encrypting 0 instead of the actual message.

- Hybrid 4.T is computationally indistinguishable from hybrid 5. This is because Π privately computes f, so there exists a simulator $\mathcal{S}$ for Π. The purpose for $\mathcal{S}$ is to simulate views for every party in I during the computation so that the corrupted parties in I cannot distinguish if they are interacting with a simulator or with other

parties. So, when we change the perfectly simulated messages for parties to messages from the simulated views for I using $\mathcal{S}$, we still get that no PPT adversary can distinguish these two worlds without breaking the privacy-preserving property of Π.

So, our simulator $\mathcal{S}'(I, \mathcal{F}_{\text{graphInfo}}(I), (x_i)_{i\in I}, f_I(\mathbf{x}))$, only requires local knowledge for any subset of parties and is indistinguishable from the 0 hybrid, where $\mathcal{S}'$ was identical to the real world:

$$\{\mathcal{S}'(I, \mathcal{F}_{\text{graph}}(I), (x_i)_{i\in I}, f_I(\mathbf{x})), f(\mathbf{x})\} \overset{c}{\approx} \{(\mathcal{S}'_0(I, \mathcal{F}_{\text{graphInfo}}([n]), \mathbf{x}, f(\mathbf{x}), f(\mathbf{x})\}$$
$$\equiv \{\text{VIEW}_I^{\Pi}(\mathbf{x}), \text{OUTPUT}^{\Pi}(\mathbf{x})\}$$

Therefore Π' is a topology-hiding realization of f. □

The following corollary is just a formal statement that we can get topology-hiding computation for all efficiently computable functions from THB and CPA-secure PKE.

Corollary 4.2.2. *Assume UC-secure THB and CPA-secure PKE exist. Then, for any efficiently computable function $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, there exists a PPT protocol Π' that is a topology-hiding realization of the functionality of f.*

Proof. For every efficiently computable function f, there exists an MPC protocol Π [26, 11, 13]. From theorem 4.2.1, this means there exists a protocol Π' which is the topology-hiding realization of Π. □

Chapter 5

Topology Hiding Broadcast protocol for General Graphs

In this chapter, we describe how our protocol works and prove that it is complete and secure.

The protocol (see protocol 1) is composed of two phases: an aggregate (forward) phase and a decrypt (backward) phase. In the aggregate phase messages traverse a walk (an exploration sequence, see definition 2.5.2) on the graph where each of the passed-through nodes adds a fresh encryption layer and homomorphically ORs the passed message with its bit. In the decrypt phase, the walk is traced back where each node deletes the encryption layer it previously added. At the end of the backward phase, the node obtains the plaintext value of the OR of all input bits. The protocol executes simultaneous walks, locally defined at each node v with d neighbors by a sequence of permutations $\pi_t \colon [d] \to [d]$ for each round t, so that at round t of the forward phase messages received from neighbor i are forwarded to neighbor $\pi_t(i)$, and at the backward phase messages received from neighbor j are sent back to neighbor $\pi_t^{-1}(j)$.

Theorem 5.0.3 (Topology-hiding broadcast for all network topologies)**.** *If there exists an OR-homomorphic PKCRand a full collection of information-local non-interfering negl(κ)-exploration sequences of length T = poly(κ), then for any network topology graph G on n nodes, there exists a polynomial-time protocol Π that is a topology-hiding realization of*

Protocol 1 Topology-hiding broadcast for general graphs. Inputs parameters: n is the number of nodes; $\mathrm{negl}(\kappa)$ the failure probability; d_i the degree of node i; and b_i the input bit of node i. See section 2.2.2 for an explanation of notation.

procedure BROADCAST$((n, \kappa, d_i, b_i))$
  // The number of steps we take in our random walk will be T
  $T \leftarrow \kappa \cdot 8n^3$
  Generate $T \cdot d_i$ key pairs: for $t \in [T]$ and $d \in [d_i]$, generate pair $(pk_{i\to d}^{(t)}, sk_{i\to d}^{(t)}) \leftarrow \mathsf{KeyGen}(1^\kappa)$.
  Generate $T-1$ random permutations on d_i elements $\{\pi_1, \cdots, \pi_{T-1}\}$. Let π_T be the identity permutation.
  // **Aggregate phase**
  For all $d \in [d_i]$, send to neighbor d the ciphertext $[b_i]_{pk_{i\to d}^{(1)}}$ and the public key $pk_{i\to d}^{(1)}$.
  for $t = 1$ to $T-1$ **do**
    for Neighbors $d \in [d_i]$ **do**
      Wait to receive ciphertext $c_{d\to i}^{(t)}$ and public key $k_{d\to i}^{(t)}$.
      Let $d' \leftarrow \pi_t(d)$.
      Compute $k_{i\to d'}^{(t+1)} = k_{d\to i}^{(t)} \circledast pk_{i\to d'}^{(t+1)}$.
      Compute $\hat{c}_{i\to d}^{(t+1)} \leftarrow \mathsf{AddLayer}\left(c_{d\to i}^{(t)}, sk_{i\to d'}^{(t+1)}\right)$ and $[b_i]_{k_{i\to d'}^{(t+1)}}$. // ciphertext under key $k_{i\to d'}^{(t+1)}$
      Compute $c_{i\to d'}^{(t+1)} \leftarrow \mathsf{HomOR}\left([b_i]_{k_{i\to d'}^{(t+1)}}, \hat{c}_{i\to d'}^{(t+1)}\right)$.
      Send $c_{i\to d'}^{(t+1)}$ and $k_{i\to d'}^{(t+1)}$ to neighbor d'.
    end for
  end for
  Wait to receive $c_{d\to i}^{(T)}$ and $k_{d\to i}^{(T)}$ from each neighbor $d \in [d_i]$.
  Compute $[b_i]_{k_{d\to i}^{(T)}}$ and let $e_{d\to i}^{(T)} \leftarrow \mathsf{HomOR}\left(c_{d\to i}^{(T)}, [b_i]_{k_{d\to i}^{(T)}}\right)$
  // **Decrypt phase**
  for $t = T$ to 1 **do**
    For each $d \in [d_i]$, send $e_{i\to d'}^{(t)}$ to $d' = \pi_t^{-1}(d)$. // Passing back
    for $d \in [d_i]$ **do**
      Wait to receive $e_{d\to i}^{(t)}$ from neighbor d.
      Compute $d' \leftarrow \pi_t^{-1}(d)$.
      $e_{i\to d'}^{(t-1)} \leftarrow \mathsf{DelLayer}\left(e_{d\to i}^{(t)}, sk_{i\to d'}^{(t)}\right)$ // If $t = 1$, DelLayer decrypts.
    end for
  end for
  // **Produce output bit**
  $b \leftarrow \bigvee_{d\in[d_i]} e_{i\to d}^{(0)}$.
  Output b.
end procedure

broadcast functionality $\mathcal{F}_{broadcast}$.

Proof. Will will show that protocol 1 is the topology-hiding realization of $\mathcal{F}_{broadcast}$. Since we assume existence of an OR-homomorphic PKCR, we are able to run our protocol. The rest of this proof is simply combining the results of lemma 5.1.1 and lemma 5.2.1.

To show protocol 1 is complete, lemma 5.1.1 states that for our parameter κ, protocol 1 outputs the correct bit for every node with probability at least $1 - \text{negl}(\kappa)$. This means, our protocol is correct with overwhelming probability with respect to the security parameter κ.

To show our protocol is sound, lemma 5.2.1 states that for our input parameter κ, an adversary can distinguish a simulated transcript from a real transcript with probability negligible in κ. Therefore, protocol 1 is sound against all PPT adversaries: they have only a negligible chance with respect to κ of distinguishing the simulation versus a real instantiation of the protocol. □

In sections section 2.5.1 and 2.5.2, we demonstrate two ways that we can construct a non-interfering, full collection of δ-exploration sequences (one with negligible δ and the other with $\delta = 0$): correlated random walks, and UESs respectively. So, we get the following two corollaries for free from the main theorem.

The first simply states that since we get an OR-homomorphic PKCRencryption scheme from ElGamal using correlated random walks. We could also use UESs, but since the analysis is less clear for UESs, we have included discussion of them in the next corollary.

Corollary 5.0.4. *There exists* $2 \cdot (\kappa \cdot 8n^3)$*-round topology-hiding broadcast for any graph* G *that succeeds with probability* $1 - negl(\kappa)$.

Proof. Lemma 2.5.8 shows that with a collection of correlated random walks of length $\kappa \cdot 8n^3$, we get $2^{-\kappa}$-exploration sequences. By theorem 5.0.3, we get a $2 \cdot \kappa \cdot 8n^3$-round topology-hiding broadcast from protocol 1 (we have to go forward through the walk and then back, hence the extra factor of 2). □

There are two sources of error in the construction in corollary 5.0.4: OR'ing Homomorphically using our ElGamal construction may fail with negligible (but non-zero) probability, and some of the random walks could fail to cover the entire graph. We can get rid

of this second source of error if instead of using random walks, we use UESs. So, if we had a perfectly complete OR-Homomorphic PKCR, then we could get a perfectly complete topology-hiding broadcast protocol.

Corollary 5.0.5. *If there exists an OR-Homomorphic PKCR without (even negligible) error, then there exists a polynomial-round topology-hiding broadcast for any graph G that always succeeds.*

Proof. Assume the existence of an OR-Homomorphic PKCRwith no error – decrypting an encrypted bit will *always* result in the bit, even after OR'ing many bits together. Note that ElGamal does not realize this because OR'ing bits has a negligible (but non-zero) chance of flipping an encrypted 1 to an encrypted 0.

Now let us analyze each walk. Every walk hits every single node in the graph because it follows a UES. So, the bit produced by every walk is going to be the OR of every node's bit, including the broadcaster's. Since there is no error in OR'ing bits together, this is guaranteed from lemma 5.1.1. So, every walk results in the output bit, and hence every party gets the output bit, so protocol 1 is perfectly complete. □

5.1 Proof of completeness

Lemma 5.1.1. *Given a full collection of non-interfering, information-local, δ-Exploration Sequences for any $\delta = negl(\kappa)$ of length T, Protocol 1 is complete; by the end of the protocol, every node gets the output bit with all but negligible probability in the security parameter κ.*

Proof. Consider one sequence, or walk, in the collection of exploration sequences. We will prove that by the end of our protocol, every node along the sequence OR's its bit and the resulting bit is decrypted. Then, we will prove that with all but probability $n \cdot \delta = \text{negl}(\kappa)$, every node has some walk that gets the output bit, meaning that with high probability, the bit b at the end of the protocol is the output bit received by each node.

So, consider a single node, u_0, with bit b_0. In the protocol, u_0's neighbors are identified by pseudonyms: u_0 just numbers them 1 to d_{u_0} and identifies them that way. We will follow

one sequence that starts at u_0 with bit b_0; u_i will identify the ith node in the sequence. For the sake of notation, pk_i will denote the public key generated by node u_i at step $i+1$ for node u_{i+1} (so $pk_i = pk^{(i+1)}_{u_i \to u_{i+1}}$), and k_i will be the aggregate key-product at step i (so $k_i = pk_0 \circledast \ldots \circledast pk_i$).

- On the first step, u_0 encrypts b_0 with pk_0 into c_1 and sends it and public key pk_0 to one of its neighbors, u_1. We will follow c_1 on its walk through T nodes.

- At step $i \in [T-1]$, c_i was just sent to u_i from u_{i-1} and is encrypted under the product $k_{i-1} = pk_0 \circledast pk_1 \circledast \cdots \circledast pk_{i-1}$, also sent to u_i. u_i computes the new public key $pk_0 \circledast \cdots \circledast pk_i = k_i$, adding its own public key to the product, encrypts b_i under k_i, and re-encrypts c_i under k_i via $\mathsf{AddLayer}$. Then, using the homomorphic OR, u_i computes c_{i+1} encrypted under k_i. u_i sends c_{i+1} and k_i to $u_{i+1} = \pi_i^{(u_i)}(u_{i-1})$.

- At step T, node u_T receives c_T, which is the encryption of $b_0 \vee b_1 \vee \cdots b_{T-1}$ under key $pk_0 \circledast \cdots \circledast pk_{T-1} = k_{T-1}$. u_T encrypts and then OR's his own bit to get ciphertext $e_T = \mathsf{HomOR}(c_T, [b_T]_{k_{T-1}})$. u_T sends e_T back to u_{T-1}.

- Now, on its way back in the decrypt phase, for each step $i \in [T-1]$, u_i has just received e_i from node u_{i+1} encrypted under $pk_1 \circledast \cdots \circledast pk_i = k_i$. u_i deletes the key layer pk_i to get k_{i-1} and then using $\mathsf{DelLayer}$, removes that key from encrypting e_i to get e_{i-1}. u_i sends e_{i-1} and k_{i-1} to $u_{i-1} = (\pi_i^{(u_i)})^{-1}(u_{i+1})$.

- Finally, node u_0 receives e_0 encrypted only under public key pk_0 on step 1. u_0 deletes that layer pk_0, revealing $e_0 = b_0 \vee \cdots \vee b_T$.

Now notice that each of these "messages" sent from every node to every neighbor follows an exploration sequence that covers the graph with probability $1 - \delta$. Let S_u be a random variable denoting the set of nodes covered by the representative sequence starting at vertex u — although $\deg(u)$ sequences start at node u, we only need to consider one of these sequences for the proof of completeness. We know that the individual probability of each of these sequences succeeding in covering the graph is $1 - \delta = 1 - \mathrm{negl}(\kappa)$, and so the probability that there exists a node whose representative sequence does *not* cover the graph

is

$$\Pr_S[\exists u : S_u \neq V] \leq \sum_{u \in V} \Pr_{S_u}[S_u \neq V] \leq n \cdot \delta = n \cdot \text{negl}(\kappa) = \text{negl}(\kappa)$$

because $n = \text{poly}(\kappa)$, and where the probability is taken over the random coins used in determining the sequences. □

5.2 Proof of soundness

We now turn to analyzing the security of our protocol, with respect to the topology-hiding security from definition 3.0.12.

Lemma 5.2.1. *If the underlying encryption OR-homomorhpic PKCR scheme is CPA-secure and a full collection of non-interfering, information-local, δ-Exploration Sequences for any $\delta = negl(\kappa)$ of length T, then protocol 1 realizes the functionality of $\mathcal{F}_{Broadcast}$ in a topology-hiding way against a statically corrupting, semi-honest adversary.*

Proof. First, we will describe an ideal-world simulator $\mathcal{S}$: $\mathcal{S}$ lives in a world where all honest parties are dummy parties and has no information on the topology of the graph other than what a potential adversary knows. More formally, $\mathcal{S}$ works as follows

1. Let Q be the set of parties corrupted by $\mathcal{A}$. $\mathcal{A}$ is a static adversary, so Q and the inputs of parties in Q must be fixed by the start of the protocol.

2. $\mathcal{S}$ sends the input for all parties in Q to the broadcast function $\mathcal{F}_{broadcast}$. $\mathcal{F}_{broadcast}$ outputs bit b_{out} and sends it to $\mathcal{S}$. Note $\mathcal{S}$ only requires knowledge of Q's inputs and the output of $\mathcal{F}_{\text{Broadcast}}$.

3. $\mathcal{S}$ gets the local neighborhood for each $P \in Q$: $\mathcal{S}$ knows how many neighbors each P has and if that neighbor is also in Q, but doesn't need to know anything else about the topology [1].

4. Consider every party $P \in Q$ such $\mathcal{N}(P) \not\subset Q$. $\mathcal{S}$ will need to simulate these neighbors not in Q.

[1] Recall that from definition 3.0.12, $\mathcal{F}_{\text{graphInfo}}$ does not reveal if nodes in Q have neighbors in common. All $\mathcal{S}$ needs to know is which neighbors are also in Q.

- **Simulating messages from honest parties in Aggregate phase.** For every $Q \in \mathcal{N}(P)$ and $Q \notin \mathcal{Q}$, $\mathcal{S}$ simulates Q as follows. At the start of the algorithm, $\mathcal{S}$ creates T key pairs:

$$(pk^{(1)}_{Q\to P}, sk^{(1)}_{Q\to P}), \cdots, (pk^{(T)}_{Q\to P}, sk^{(T)}_{Q\to P}) \leftarrow \mathsf{Gen}(1^\kappa)$$

  At step $t = i$ in the for loop on line 8, $\mathcal{S}$ simulates Q sending a message to P by sending $([0]_{pk^{(i)}_{Q\to P}}, pk^{(i)}_{Q\to P})$. $\mathcal{S}$ receives the pair $(c^{(i)}_{P\to Q}, k^{(i)}_{P\to Q})$ from P at this step.

- **Simulating messages from honest parties in the Decrypt phase.** Again, for every $P \in \mathcal{Q}$, $Q \in \mathcal{N}(P)$ and $Q \notin \mathcal{Q}$, $\mathcal{S}$ simulates Q. At $t = i$ in the for loop on line 20, $\mathcal{S}$ sends $[b_{out}]_{k^{(i)}_{Q\to P}}$ to P. $\mathcal{S}$ receives $e^{(i)}_{P\to Q}$ from P.

We will prove that any PPT adversary cannot distinguish whether he is interacting with the simulator $\mathcal{S}$ or with the real network except with negligible probability.

(a) Hybrid 1. $\mathcal{S}$ simulates the real world exactly and has information on the entire topology of the graph, each party's input, and can simulate each sequence identically to how the walk would take place in the real world.

(b) Hybrid 2. $\mathcal{S}$ replaces the real keys with simulated public keys, but still knows everything about the graph (as in Hybrid 1). Formally, for every honest Q that is a neighbor to $P \in \mathcal{Q}$, $\mathcal{S}$ generates

$$(pk^{(1)}_{Q\to P}, sk^{(1)}_{Q\to P}), \cdots, (pk^{(T)}_{Q\to P}, sk^{(T)}_{Q\to P}) \leftarrow \mathsf{Gen}(1^\kappa)$$

and instead of adding a layer to the encrypted $[b]_{pk^*}$ that P has at step t, as done in line 12 and 13, $\mathcal{S}$ computes $b' \leftarrow b_Q \vee b$ and sends $[b']_{pk^{(i)}_{Q\to P}}$ to P during the aggregate phase; it is the same message encrypted in Hybrid 1, but it is now encrypted under an unlayered, fresh public key. In the decrypt phase, each honest Q neighbor to P will get back the bit we get from the sequence of OR's encrypted under that new public key as well; the way all nodes in $\mathcal{Q}$ peel off layers of keys guarantees this will still be correct.

(c) Hybrid 3. $\mathcal{S}$ now simulates the ideal functionality during the aggregate phase, sending encryptions of 0. Formally, during the aggregate phase, every honest Q that is a neighbor to $P \in Q$ $\mathcal{S}$ sends $[0]_{pk^{(i)}_{Q \to P}}$ to P instead of sending $[b']_{pk^{(i)}_{Q \to P}}$. Nothing changes during the decrypt phase; the simulator still sends the resulting bit from each sequence back and is not yet simulating the ideal functionality.

(d) Hybrid 4. $\mathcal{S}$ finally simulates the ideal functionality at the during the decrypt phase, sending encryptions of b_{out}, the output of $\mathcal{F}_{\text{Broadcast}}$, under the simulated public keys. This is instead of simulating the sequences through the graph and ORing only specific bits together. Notice that this hybrid is equivalent to our original description of $\mathcal{S}$ and requires no knowledge of other parties' values or of the graph topology other than local information about Q (as specified by the $\mathcal{F}_{\text{graphInfo}}$ functionality).

Now, let's say we have an adversary $\mathcal{A}$ that can distinguish between the real world and the simulator. This means $\mathcal{A}$ can distinguish between Hybrid 1 and Hybrid 4. So, $\mathcal{A}$ can distinguish, with non-negligible probability, between two consecutive hybrids. We will argue that given the security of our public key scheme and the high probability of success of the algorithm, that this should be impossible.

(a) First, we claim no adversary can distinguish between Hybrid 1 and 2. The difference between these Hybrids is distinguishing between AddLayer and computing a fresh encryption key. In Hybrid 1, we compute a public key sequence, multiplying public key k by a freshly generated public key. In Hybrid 2, we just use a fresh public key. Recall that the public keys in our scheme form a group. Since the key sequence $k \circledast pk_{new}$ has a new public key that has not been included anywhere else in the transcript, $k \circledast pk_{new}$ can be thought of as choosing a new public key independently at random from k. This is the same distribution as just choosing a new public key: $\{k \circledast pk_{new}\} \equiv \{pk_{new}\}$. Therefore, any tuple of multiplied keys and fresh keys are indistinguishable from each other. So, no adversary $\mathcal{A}$ can distinguish between Hybrids 1 and 2.

(b) Now we will show that no PPT adversary can distinguish between Hybrids 2

and 3. The only difference between these two hybrids is that $\mathcal{A}$ sees encryptions of the broadcast bit as it is being transmitted as opposed to seeing only encryptions of 0 from the simulator. Note that the simulator chooses a key independent of any key chosen by parties in Q in each of the aggregate rounds, and so the bit is encrypted under a key that $\mathcal{A}$ does not know. This means that if $\mathcal{A}$ can distinguish between these two hybrids, then $\mathcal{A}$ can break semantic security of the scheme, distinguishing between encryptions of 0 and 1.

(c) For this last case, we will show that there should not exist a PPT adversary $\mathcal{A}$ that can distinguish between Hybrids 3 and 4.

There are two differences between Hybrids 3 and 4. The first is that for each sequence S, during the decrypt phase, we send $b_{out} = \bigvee_{i\in[n]} b_i$, the OR of all of the node's bits, instead of $b_S = \bigvee_{u\in S} b_u$, the OR of all node's bits in that specific length-T sequence.

Recall that each sequence has probability at least $1 - \text{negl}(\kappa) = 1 - \delta$ of covering the graph. There are two sequences starting at each edge, making for at most $2n^2$ simultaneous sequences. So, the probability that there exists a sequence S so that $b_{out} \neq b_S$ is at most $2n^2 \cdot \text{negl}(\kappa) = \text{negl}(\kappa)$ by a union bound. Therefore, this difference in hybrids is undetectable to any polynomial adversary.

The second difference is that our simulated encryption of b_{out} is generated by making a fresh encryption of b_{out}. But, if $b_{out} = b_S$ (which it will with overwhelming probability), by the claim 2.4.1, the encryption generated by ORing many times in the graph is computationally indistinguishable to a fresh encryption of b_{out}. Therefore, computationally, it is impossible to distinguish between Hybrids 3 and 4.

□

5.3 Complexity and Optimizations

Note that we have two methods for realizing Protocol 1: correlated random walks and UES. In this section, we will give upper bounds on the communication complexity under the random walk instantiation of protocol 1 and discuss optimizations for graph families where tighter cover time bounds are known. We will then discuss communication complexity when using UES.

5.3.1 Communication Complexity with Correlated Random Walks

We show that the communication complexity is $\Theta(B\kappa m)$ group elements, where B is an upper bound on the cover time of the graph (for our protocol on general graphs, we have $B = 4n^3$). We measure the communication complexity in terms of the overall number of group elements transmitted throughout the protocol (where the group elements are for the ciphertext and public-key pairs of the underlying DDH-based encryption scheme, and their size is polynomial in the security parameter).

Claim 5.3.1 (Communication complexity). *The communication complexity of protocol 1 using correlated random walks of length $T = 2\kappa B$ is $\Theta(B\kappa m)$ group elements.*

Proof. The random-walks in protocol 1 are of length $T = 2B\kappa$, yielding $2T$ total rounds of communication including both the forward and backwards phases. At each round, every node v sends out $\deg(v)$ messages. Summing over all $v \in V$, all of the nodes communicate $2m$ messages every round – one for each direction of each edge (for m denoting the number of edges in the network graph). By the end of the protocol, the total communication is $4Tm = \Theta(B\kappa m)$. □

We conclude the communication complexity of protocol 1 on input n, κ is $\Theta(\kappa n^5)$ group elements.

Corollary 5.3.2. *On input n, κ, the communication complexity of protocol 1 is $\Theta(\kappa n^5)$ group elements.*

Type of Graph	Cover time
Arbitrary graph [21]	$O(n^3)$
Expanders [5]	$O(n \log n)$
Regular Graphs [19]	$O(n^2)$

Table 5.1: Cover times for specific graphs.

Proof. For a graph with at most n nodes, $B = 4n^3$ is an upper bound on the cover time (see theorem 2.5.6), and $m = n^2$ is an upper bound on the number of edges. Assigning those B, m in the bound from claim 5.3.1, the proof follows: $\Theta(B\kappa m) = \Theta(\kappa \cdot n^3 \cdot n^2) = \Theta(\kappa n^5)$. □

Better Bounds on Cover Time for Some Graphs

Now that we have seen how the cover time bound B controls both the communication and the round complexity, we will look at how to get a better bound than $O(n^3)$.

Cover time has been studied for various kinds of graphs, and so if we leak the kind of graph we are in (e.g. expanders), then we can use a better upper bound on the cover time, shown in table 5.1.

For example on expander graphs (arising for example in natural applications on random regular graphs), it is known that the cover times $C_G = O(n \log n)$, much less than $O(n^3)$ [5]. This means that for expanders, we can run in $C_G = O(n \log n)$ round complexity, and $O(C_G \kappa m) = O(\kappa mn \log n)$ communication complexity. Even assigning the worst case bound $m \leq n^2$, we get round and communication complexity $O(n \log n)$ and $O(\kappa n^3 \log n)$ respectively—much better than the general case that has $O(\kappa n^3)$ round complexity and $O(\kappa n^5)$ communication complexity.

5.3.2 Communication Complexity with Universal Exploration Sequences

Unfortunately, we are less precise when discussing communication complexity of our protocol when using UESs. This is because known explicit, deterministic, polynomial-time constructions use *log-space*, and these works, as far as we could find, do not discuss how long the resulting sequence is. Moreover, every source with the exception of Koucky's thesis only discusses d-regular graphs[20] . Koucky's work provides a transformation of

d-regular graph sequences to general graphs at a cost which requires knowing the number of edges in the original graph.[2] With that in mind, we will discuss what is known and, if advice is allowed to be given to nodes, about how long these sequences may need to be.

Reingold's paper implies that the algorithm for computing the exploration sequences for general graphs is *log space*, meaning the running time of computing such an exploration sequence and the length of the resulting sequence could be anything polynomial. However, looking at Koucky's thesis, we can get a generic transformation of a *universal traversal sequence* (UTS) on a regular graph of length T to one of length approximately $O(n^2 \cdot T)$.

Theorem 5.3.3 ([25]). *There exists a log-space algorithm that takes as input 1^n and outputs a universal traversal sequence on 3-regular graphs with n nodes.*

Log-space implies polynomial time and polynomial length in n, the number of nodes. So, what is left is to be able to transform a UTS on a 3-regular graph to a UTS on general graphs if we only know the number of nodes. For this transformation, we will rely on the following theorem from Koucky's thesis [20].

Theorem 5.3.4 ([20]). *Let $m \geq 1$ be an integer. For any traversal sequence $\tau_1, \ldots, \tau_t$, that is universal for 3-regular graphs on $3m$ vertices, we can compute, with AC^0 circuits, an exploration sequence that is universal for graphs containing m edges.*

Lemma 5.3.5. *We can produce a UES on general graphs with n nodes of length $O(n^2 \cdot T)$ where T is the maximum length of a UTS generated by Reingold's algorithm for 3-regular graphs with $3(n-1)$ to $3n^2$ nodes.*

Proof. We will essentially be applying theorem 5.3.3 in conjunction with theorem 5.3.4 $O(n^2)$ times because we do not know the exact number of edges in our graph.

Let $S = ()$ be an empty sequence. For every $m \in \{n-1, \ldots, n^2\}$, we will use Reingold's algorithm to construct a UTS for 3-regular graphs on $3m$ nodes, and then transform it into a UES on general graphs with m edges. We then append this sequence to S.

Now, for any connected graph on n vertices, it will have somewhere between $n-1$ and n^2 edges (to be connected). Let m^* be the number of edges it has. There is some subse-

[2]The transformation actually takes universal *traversal* sequences on d-regular graphs and turns them into universal exploration sequences on general graphs with $m = 3d$ edges

quence in S that is a UES for general graphs on n nodes with m^* edges; that subsequence is guaranteed to explore the graph.

S has a length equal to the sum of all of the exploration sequences for each m. There are $O(n^2)$ such m's, and so we can upper bound the total length using the longest such exploration sequence (length T): the length of S is $O(n^2T)$. □

It is interesting to note that since UESs are guaranteed to cover the graph, their length does not depend at all on the security parameter κ, unlike the random walk construction. Therefore, it is more efficient to use UESs in this protocol if n is small compared to κ. However, since we do not have good bounds on how long these constructable UESs are, we cannot give the exact point at which it becomes better to use this method.

Chapter 6

Conclusion and Future Work

This work showed that topology-hiding computation is feasible for *every* network topology (in the computational setting, assuming DDH), using random walks or UESs. This resolution completes a line of works on the feasibility of topology hiding computation against a static semi-honest adversary [22, 18, 1]. It leaves open the feasibility question against a malicious or adaptive adversary.

Although there are impossibility results for even very weak malicious adversaries (fail-stop) [22], the adversary is able to learn about the topology of the graph because it is able to disconnect it. So, if we first limit the adversary so that it cannot disconnect the graph, there is hope that we can get some results (say a $(t + 1)$-connected graph and the adversary can abort/control at most t nodes).

Then there is the model of dynamic graphs, which are especially relevant in some mesh networks. For example, consider the following application: smart cars on a highway communicating with their local neighbors about weather, traffic, and other hazards, without needing to coordinate their information with a third party or reveal their location relative ot other vehicles. Cars are constantly entering and exiting the highway and changing location relative to other cars, so the graph, while it remains connected, is not static. If we consider a passive, static adversary, then we can still get around the impossibility result. Perhaps we can even adjust this relatively simple protocol to work for these kinds of graphs.

Finally, there is the question of what other cryptosystems are OR-homomorphic PKCR encryption schemes. For example, assuming that quadratic residues are difficult to dis-

tinguish from non-resides (the QR assumption), we can get a PKCRencryption scheme (this is via Cock's IBE [7], but not shown in this thesis). However, this scheme is only XOR-homomorphic. Lattice-based cryptography may also be viable. Showing that other assumptions can achieve topology-hiding computation strengthens the result, and in the case of lattice-based cryptography, we get post-quantum security.

Topology hiding computation is a relatively unexplored subject in cryptography, having (in the computational setting) its first feasibility result in 2015 [22]. It will be exciting to see what else can be proved in this model.

Bibliography

[1] Adi Akavia and Tal Moran. Topology hiding computation beyond logarithmic diameter. In *To appear in Eurocrypt*, 2017.

[2] József Balogh, Béla Bollobás, Michael Krivelevich, Tobias Müller, and Mark Walters. Hamilton cycles in random geometric graphs. *The Annals of Applied Probability*, 21(3):1053–1072, 2011.

[3] Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 387–404. Springer, 2014.

[4] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[5] A. K. Chandra, P. Raghavan, W. L. Ruzzo, and R. Smolensky. The electrical resistance of a graph captures its commute and cover times. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 574–586, New York, NY, USA, 1989. ACM.

[6] Nishanth Chandran, Wutichai Chongchitmate, Juan A. Garay, Shafi Goldwasser, Rafail Ostrovsky, and Vassilis Zikas. The hidden graph model: Communication locality and optimal resiliency with adaptive faults. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, pages 153–162, New York, NY, USA, 2015. ACM.

[7] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings*, pages 360–363, 2001.

[8] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *J. ACM*, 40(1):17–47, January 1993.

[9] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 263–270. ACM, 1999.

[10] Tobias Friedrich, Thomas Sauerwald, and Alexandre Stauffer. Diameter and broadcast time of random geometric graphs in arbitrary dimensions. *Algorithmica*, 67(1):65–88, 2013.

[11] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[12] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, New York, NY, USA, 2004.

[13] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 578–602. Springer, 2014.

[15] Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, pages 157–168, New York, NY, USA, 2016. ACM.

[16] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 132–150. Springer, 2011.

[17] Markus Hinkelmann and Andreas Jakoby. Communications in unknown networks: Preserving the secret of topology. *Theoretical Computer Science*, 384(2–3):184–200, 2007. Structural Information and Communication Complexity (SIROCCO 2005).

[18] Martin Hirt, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Network-hiding communication and applications to multi-party protocols. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 335–365, 2016.

[19] Jeff D Kahn, Nathan Linial, Noam Nisan, and Michael E Saks. On the cover time of random walks on graphs. *Journal of Theoretical Probability*, 2(1):121–128, 1989.

[20] Michal Koucky. *On Traversal Sequences, Exploration Sequences and Completeness of Kolmogorov Random Strings*. PhD thesis, New Brunswick, NJ, USA, 2003. AAI3092958.

[21] Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[22] Tal Moran, Ilan Orlov, and Silas Richelson. Topology-hiding computation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015*, volume 9014 of *Lecture Notes in Computer Science*, pages 169–198. Springer, 2015.

[23] Mathew Penrose. *Random geometric graphs*. Number 5. Oxford University Press, 2003.

[24] Gregory J Pottie and William J Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[25] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008.

[26] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.