# *Topology-Hiding Computation on All Graphs*

# Topology-Hiding Computation on all Graphs

**Cite this article as:** Adi Akavia, Rio LaVigne and Tal Moran, Topology-Hiding Computation on all Graphs, Journal of Cryptology https://doi.org/10.1007/s00145-019-09318-y

# Topology-Hiding Computation on all Graphs

Adi Akavia* Rio LaVigne† Tal Moran‡

March 4, 2019

**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 either the Decisional Diffie-Hellman or Quadratic-Residuosity 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).



## 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 [36, 16, 15]. 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. Moreover, in some cases 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 [22], 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 [30]; followed by constructions with improved efficiency that make only black-box use of underlying primitives

*The Academic College of Tel-Aviv Jaffa. Email: akavia@mta.ac.il. Work partly supported by the ERC under the EU's Seventh Framework Programme (FP/2007-2013) ERC Grant Agreement no. 307952.

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

‡IDC Herzliya. Email: talm@idc.ac.il. Supported by ISF grant no. 1790/13.

[23]. However, all these protocols 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 [2] 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 [30, 23, 2]. They include, for example, wireless and ad-hoc sensor networks (e.g. mesh networks for cellphones, etc), as in [12, 32]. The topology in these graphs is modeled by random geometric graphs [31], where, with high probability, the diameter and the circumference are simultaneously large [13, 5]. 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 (and similarly under the Quadratic Residuosity (QR) assumption), thus positively resolving the above open problem. The adversary is static, semi-honest, and can corrupt any number of parties, as in the prior works [30, 23, 2].[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. It is important to note that these prior protocols also work within this model, however, it is not until this work that we formally define the pseudonym model as a stronger model.

**Theorem 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 or QR assumption, and provided the parties are given an upper-bound n on the number of nodes).*

The formal theorem is stated and proved as theorem 5.4.

As in [30, 23, 2], 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. More explicitly, broadcast plus a public-key interface implies that we can simulate point-to-point channels, as detailed in Appendix B. Then, put together with the existence of secure MPC for all efficiently computable functionalities (assuming parties have access to a broadcast channel and that public key encryption exists) [36, 16, 15], we conclude that *topology-hiding MPC* for all efficiently computable functionalities and all network topologies exists (assuming public key encryption exists).

**Corollary 1.2.** *There exists a topology-hiding protocol realizing any efficiently computable functionality on every network topology (under DDH or QR assumption, and provided parties are given an upper-bound n on the number of nodes).*

This is stated more formally as theorem 5.5, and the proof that broadcast implies general computation is formally stated and proved in the UC model in Appendix B.

## 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 [2]. 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.

[1] Moran et al.[30] consider also a fail-stop adversary for proving an impossibility result.

A challenge we face is that the random 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, first recall that the functionality of broadcast is as follows: a source party $i$ has a source-bit $b_i$ and by the end of the broadcast, all parties learn $b_i$. Next, observe that broadcast can be realized from computing the OR of all input bits, where the source party uses $b_i$ as its input and all other parties input 0. Once we have broadcast, there is a simple method using a public-key cryptosystem for compiling broadcast into an multiparty functionality. Details on this are given in Section B.2 and proved formally in the UC model in Appendix B.

Now, let us 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 removes 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 note that in our protocol, we will not require homomorphic properties from our encryption scheme: computing OR "homomorphically" on an encrypted bit $c$ and a known bit $b$ is simply $c$ if $b = 0$ or an encryption of 1 if $b = 1$. This is an improvement over the requirements of previous schemes.

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 $\pi_t \colon [d] \to [d]$, 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 all of our walks traverse the graph, by 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 Works

The related work considering the multi-party perspective can be divided into roughly four categories: previous work done in the computational setting, previous work in the information-theoretic setting,

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

| **Graphs families** | [23, 30] | [2] | [This Work] |
|---|---|---|---|
| Log diameter constant degree | + | − | + |
| Cycles, trees | − | + | + |
| Log circumference | − | + | + |
| Log diameter super-constant degree | − | − | + |
| Regular graphs | − | − | + |
| Arbitrary graphs | − | − | + |

Table 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.

secure MPC in general (incomplete) networks, and finally, beyond the multi-party perspective, there is related work dealing with the kind of tools that we use (our PKCR encryption, defined in Section 3.1).

**Topology Hiding in Computational Settings.** Table 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 in the work of Moran, Orlov, and Richelson [30]. 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 much more similar to the recent Akavia-Moran paper [2], 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 arbitrary graphs instead of deterministically figuring out the path beforehand.

Other related works include a work by Hirt, Maurer, Tschudi and Zikas [23], which describes a protocol that achieves better efficiency than [30] (and does not require general secure computation), but is still restricted to network graphs with logarithmic diameter. Addressing a problem different from topology-hiding, the work by Chandran, Chongchitmate, Garay, Goldwasser, Ostrovsky, and Zikas [9] reduces communication complexity of secure MPC by allowing each party to communicate with a small (sublinear in the number of parties) number of its neighbors. And finally, since the publication of the original CRYPTO paper [1], there have been a couple of works extending these ideas to a stronger adversarial setting: fail-stop. Here, the adversary is allowed to abort nodes, and while there is an impossibility result for getting any topology-hiding broadcast against this adversary, if a small amount of leakage is allowed (less than 1 bit), then we can get around the impossibility. First, Ball, Boyle, Malkin, and Moran were able to accomplish this assuming secure hardware [4]. Later, LaVigne, Liu, Maurer, Mularczyk, and Moran were able to get this same result from the standard assumptions [28]. As an additional result, they were able to show how to get this mixing kind of encryption (privately key-commutative randomizeable) from Learning With Errors. Previously it had only been known from Decision Diffie-Hellman and Quadratic Residuosity.

**Topology Hiding in Information-Theoretic Settings.** Hinkelmann and Jakoby [22] 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.** Halevi, Ishai, Jain, Kushilevitz, and Rabin [19] presented a unified framework for studying secure MPC with arbitrarily restricted interaction patterns, generalizing models for MPC with specific restricted interaction patterns [17, 6, 20]. 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 $x_i, \ldots, x_n$. This model allows complete knowledge of the underlying interaction pattern (or as in our case, the communication graph).

**Layered Public-Key Encryption** Our techniques rely on something defined in the Akavia-Moran work called Privately Key-Commutative Randomizable Encryption (PKCR encryption) [2]. The main idea behind this tool is that one can *change* (or layer) the public key encrypting a message with ones own secret key, and then later undo that change with the secret key. Moreover, one must be able to re-randomize the encryption given the public key.

Halevi, Lindell, and Pinkas also explored the idea of having a layered encryption, where parties add and remove layers in a very explicit sense (re-encrypting an encrypted message), in 2011 [21]. They call this "layer re-randomizable encryption." In their work, they encrypt a message with a vector of public keys, and allow for re-randomization of ciphertexts when encrypted with these layers. Then, Gordon, Malkin, Rosulek, and Wee generalized layer re-randomizable encryption in 2013 [18]. Instead of having a vector of public keys, they aggregate the vector of public keys more formally, treating the resulting aggregate key as a "normal" public key. This work is probably the closest that comes to what we need from PKCR encryption, however notice that in all of these prior works, the layers of public keys are not hidden, or if they are, this is not an inherent property of the scheme.

One of the most important qualities we need from our encryption scheme is to hide what the public key was before adding a layer: if one do not know anything about the public-secret key-pair that was used to change the public key encrypting a certain message, one cannot reconstruct the original public key.

Along this line, Hirt et. al. accomplish this with their primitive, "multi-homomorhpic threshold encryption with reversible randomization" [23]. This complex object consists of many parts: threshold encryption (decryption involves decrypting with shares), and randomizing the public key that encrypts a message. This primitive allows them to get away with a protocol that requires relatively few rounds. To contrast, we only need the key and message randomization aspects. To that end, [2] defined their own, simpler primitive: privately key-commutative randomizable encryption. This simple and intuitive primitive is what we are able to use in our work.

## 1.4 Organization of Paper

In sections 2 to 3 we describe our adversarial model and introduce definitions and our notation. In section 3.1 we detail the special properties we require from the encryption scheme that we use in the cycle protocol, and show how it can be instantiated based on DDH and QR. In section 3.2, 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 4, we define our security model, which is slightly stronger than the one in both the work of Akavia and Moran and Hirt et al.[23, 2]. In section 5, we explain our protocol for topology-hiding broadcast on general graphs and prove its completeness and security, going over a time

and communication tradeoff and explaining how we can optimize our protocol with respect to certain classes of graphs. Finally, in section 6, we conclude and discuss future work.

# 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 $\text{poly}(\kappa)$ and $\kappa$ is the security parameter. We identify $V$ with the set of parties $\{P_1, \ldots, P_n\}$.

We consider a static, passive and computationally bounded (PPT) adversary that corrupts some subset of parties (any number of parties). That is, at the beginning of the protocol, the adversary corrupts a subset of the parties may see all communication that passes through them. However, because the adversary is passive, its corrupted parties may not deviate from the protocol. For general MPC definitions including in-depth descriptions of the adversarial models we consider, see [14].

## 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 $d_v = |\mathcal{N}(v)|$. Notice that $\mathcal{N}(v)$ does *not* contain $v$.

### 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 $pk^{(t)}_{i\to d}$ represent the public key created by node $i$ to be used for neighbor $d$ at round $t$; $sk^{(t)}_{i\to d}$ is the corresponding secret key. Ciphertexts are labeled similarly: $c^{(t)}_{d\to i}$, is from neighbor $d$ to node $i$ (encrypted under $pk^{(t)}_{i\to d}$).

## 2.3 UC Security

As in [30], we prove security in the UC model [7]. 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 [7].

# 3 Key tools

In this section we describe our two main tools to obtain topology-hiding computation. The first is from [2]: PKCR-encryption. This tool will allow every party in the network to add at least one (maybe more) of their own public keys to a message that traverses the whole graph. The second is Graph Exploration Sequences. These allow messages to traverse the whole graph with, being sent from one node to one of its neighbors in a locally-computable way. We will go into more detail in the following subsections.

### 3.1 Privately Key-Commutative and Randomizable Encryption

As in [2], we require a public key encryption scheme with the properties of being *privately key-commutative*, and *re-randomizable*. However, unlike all previous work using this strategy ([2], [1],[28]), we will *not* need any homomorphic properties from our encryption scheme. 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 and the Quadratic Residue assumption. It was shown in work by [28] that one can also get this property from the Learning-With-Errors assumption.

We call an encryption scheme satisfying this property, privately key-commutative and re-randomizable, a *PKCR-encryption*.

#### 3.1.1 Required Properties

Let $\mathcal{PK}$ be the space of public keys (which must form a group under the $\circledast$ operation), $\mathcal{SK}$ the space of secret keys, $\mathcal{M}$ the space of plaintext messages, and $C$ the space of ciphertexts. Because the central theme of PKCR encryption is the ability to combine public keys, we will use the notation $(\mathsf{pk}, \mathsf{sk})$ to denote a generic public-key secret-key pair, and $\mathbf{k}$ will denote a possibly combined public key; although $\mathsf{pk}$ and $\mathbf{k}$ are elements from the same set ($\mathcal{PK}$), this distinction will help with readability of the protocols ($\mathbf{k}$ generally means that no one has access to the corresponding secret key). We will use the shorthand $[m]_{\mathsf{pk}}$ to denote an encryption of the message $m$ under public-key $\mathsf{pk}$. We assume that for every secret key $\mathsf{sk} \in \mathcal{SK}$ there is associated a single public key $\mathsf{pk} \in \mathcal{PK}$ such that $(\mathsf{pk}, \mathsf{sk})$ are in the range of $\mathsf{KeyGen}$. We slightly abuse notation and denote the public key corresponding to $\mathsf{sk}$ by $\mathsf{pk}(\mathsf{sk})$.

A PKCR encryption scheme will consist of 6 functions. The first three are familiar to any public-key cryptosystem. Below are the functions we need to get PKCR; the three new functions will be covered in more detail in the following subsections.

- $\mathsf{KeyGen} : \{0,1\}^* \to \mathcal{PK} \times \mathcal{SK}$ generates public-secret key-pairs. This function may also take in system parameters (common public parameters, e.g. a modulus or group generator). There is implicit randomness involved.
- $\mathsf{Enc} : \mathcal{M} \times \mathcal{PK} \times \{0,1\}^* \to C$ encrypts a message under a public key. There is implicit randomness.
- $\mathsf{Dec} : C \times \mathcal{SK} \to \mathcal{M}$ decrypts a message encrypted under some public key using the associated secret key. Can be deterministic or random, but the probability of a successful decryption must be at least $1 - \mathrm{negl}(\kappa)$.
- $\mathsf{Rand} : C \times \mathcal{PK} \to C$ re-randomizes a ciphertext encrypted encrypted under the given public key, so that as long as the given ciphertext decrypts correctly, it outputs a ciphertext indistinguishable from a fresh encryption of the message. There is implicit randomness involved.
- $\mathsf{AddLayer} : C \times \mathcal{PK} \times \mathcal{SK} \to C$ takes an encrypted message, and then "adds a layer" to the ciphertext corresponding to the public key of the input secret key. I.e. $\mathsf{AddLayer}(c, \mathsf{pk}, \mathsf{sk})$ outpus a ciphertext distributed computationally indistinguishable (even with the secret key) from a fresh encryption of the same message under the public key $\mathbf{k}' \circledast \mathsf{pk}(\mathsf{sk})$: $\mathsf{Enc}(m, \mathbf{k}' \circledast \mathsf{pk}(\mathsf{sk}))$. There is implicit randomness involved.
- $\mathsf{DelLayer} : C \times \mathcal{PK} \times \mathcal{SK} \to C$ takes an encrypted message, and then "removes a layer" to the ciphertext corresponding to the public key of the input secret key. I.e. $\mathsf{DelLayer}(c, \mathsf{pk}, \mathsf{sk})$ is distributed computationally-indistinguishably (even with the secret key) from a fresh encryption $\mathsf{Enc}(m, \mathbf{k}' \circledast (\mathsf{pk}(\mathsf{sk}))^{-1})$. There is implicit randomness involved.

For the concrete instantiations of PKCR, we will need to define another function that generates the publicly shared parameters (protocol parameters, or system parameters). For example, if using ElGamal, all parties will need to be working in the same group with the same generator. We call this functionality $\mathsf{Gen}$, and it will never be needed to be called by a party — parties are assumed to have this basic system information at the start of the protocol.

**Privately Key-Commutative: AddLayer and DelLayer**

The set of public keys $\mathcal{PK}$ form an abelian (commutative) group. We denote the group operation $\circledast$. Given any $\mathbf{k}_1, \mathbf{k}_2 \in \mathcal{PK}$, there exists an efficient algorithm to compute $\mathbf{k}_1 \circledast \mathbf{k}_2$. We denote the inverse of $\mathbf{k}$ by $\mathbf{k}^{-1}$ (i.e. $\mathbf{k}^{-1} \circledast \mathbf{k}$ is the identity element of the group). Given a secret key $\mathsf{sk}$, there must be an efficient algorithm to compute the inverse of its public key $(\mathsf{pk}(\mathsf{sk}))^{-1}$.

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

1. For every public key $\mathbf{k}' \in \mathcal{PK}$, every secret key $\mathsf{sk} \in \mathcal{SK}$, every message $m \in \mathcal{M}$ and every ciphertext $c = [m]_{\mathbf{k}'}$,
$$\mathsf{AddLayer}(c, \mathbf{k}', \mathsf{sk}) \equiv \mathsf{Enc}(m, \mathbf{k}' \circledast \mathsf{pk}(\mathsf{sk})) .$$
2. For every public key $\mathbf{k}' \in \mathcal{PK}$, every secret key $\mathsf{sk} \in \mathcal{SK}$ every message $m \in \mathcal{M}$ and every ciphertext $c = [m]_{\mathbf{k}'}$,
$$\mathsf{DelLayer}(c, \mathbf{k}', \mathsf{sk}) \equiv \mathsf{Enc}(m, \mathbf{k}' \circledast (\mathsf{pk}(\mathsf{sk}))^{-1}) .$$

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.

Although we have defined our primitive to be commutative, our protocol will only require removing the most recently-added layer. This is because our protocol has a forward and backward phase, where the backward phase removes layers in the opposite order in which they were added. So, while commutivity is not strictly needed, the commutative-group property of keys makes it easy to argue that adding and deleting layers is equivalent to encrypting under specific keys.

**Randomizable: Rand**

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 $\mathsf{pk} \in \mathcal{PK}$ and ciphertext $c = [m]_{\mathsf{pk}}$, the distributions $(m, \mathsf{pk}, c, \mathsf{Rand}(c, \mathsf{pk}; U^*))$ and $(m, \mathsf{pk}, c, \mathsf{Enc}(m, \mathsf{pk}; U^*))$ are computationally indistinguishable.
2. *Neutrality:* For every ciphertext $c \in C$, every secret key $\mathsf{sk} \in \mathcal{SK}$ and every choice of randomness $r \in \{0,1\}^*$,
$$\mathsf{Dec}(c, \mathsf{sk}) = \mathsf{Dec}(\mathsf{Rand}(c, \mathsf{pk}(\mathsf{sk}); r), \mathsf{sk}) .$$

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 $\mathsf{KeyGen}$ is computationally indistinguishable from a fresh public-key generated by $\mathsf{KeyGen}$.

### 3.1.2 Instantiation of PKCR-enc under DDH

We use standard ElGamal, with a special method for encrypting only 0's and 1's, and augmented by the additional required functions.

- $\mathsf{EG.Gen}(1^\kappa)$. Generates master public parameters: a $\kappa$-secure DDH prime $p$ and generator $g$ of $\mathbb{Z}_p^*$. If it is clear which modulus and generator we are using from context, these public parameters are not written out as input.
- $\mathsf{EG.KeyGen}(g, p)$. Choose a secret key $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_{p-1}$ and compute the public key $h = g^{\mathsf{sk}}$. Output the secret key $\mathsf{sk}$ and public key $\mathsf{pk} = h$.

- $\mathsf{EG.Enc}(m \in 0, 1, \mathsf{pk})$. Choose $r \xleftarrow{\$} \mathbb{Z}_{p-1}$, and let $c_1 = g^r$. If $m = 0$, let $c_2 = \mathsf{pk}^r \cdot g^0$. If $m = 1$, choose another $s \xleftarrow{\$} \mathbb{Z}_{p-1}$ such that $s \neq 0$, and let $c_2 = \mathsf{pk}^r \cdot g^s$. Output $c = (c_1, c_2)$.
- $\mathsf{EG.Dec}(c = (c_1, c_2), \mathsf{sk})$. Evaluate $c_1^{-\mathsf{sk}} c_2 = g^{-\mathsf{sk} \cdot r} \cdot \mathsf{pk}^r m = m$. Output $m$.

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 $G = \mathbb{Z}_p^*$ and the generator $g$, and different public keys vary only in the element $h = g^x$. So, $g$ is always the group generator in $\mathbb{Z}_p^*$. Below, we define the Rand functionality, which is standard for ElGamal:

**function** $\mathsf{EG.Rand}(c = (c_1, c_2), \mathsf{pk}; r)$
  **return** $(c_1 \cdot g^r, \mathsf{pk}^r \cdot c_2)$
**end function**

We use the shorthand notation of writing $\mathsf{Rand}\,(c, \mathsf{pk})$ when the random coins $r$ are chosen independently at random during the execution of Rand.

**Lemma 3.1.** EG.*Rand is a randomization of ElGamal ciphertexts that fits both the randomization and neutrality constraints.*

*Proof.* First, we show that it randomizes the ciphertext. That is, we need to show that the distribution $(m, \mathsf{pk}, c, \mathsf{Rand}(c, \mathsf{pk}; U^*))$ is at least computationally indistinguishable from $(m, \mathsf{pk}, c, \mathsf{Enc}(m, \mathsf{pk}; U^*))$. We we actually show that these distributions are equivalent for any ciphertext that is an encryption of $m$.

We rewrite $c = (c_1, c_2) = (g^{r_1}, \mathsf{pk}^{r_1} \cdot m)$, and let $c' = \mathsf{Rand}(c, \mathsf{pk}; U^*) = (g^{r_1+r_2}, \mathsf{pk}^{r_1+r_2} \cdot m)$, and $\hat{c} = \mathsf{Enc}(m, \mathsf{pk}; U^*) = (g^{r'_1}, \mathsf{pk}^{r'_1} \cdot m)$. For any $r \in \mathbb{Z}_{p-1}$, $\Pr_{\mathsf{Rand}}[r_1 + r_2 = r] = \Pr_{\mathsf{Enc}}[r'_2 = r] = \frac{1}{p-1}$. So, the distribution of the exponent in both $c'$ and $\hat{c}$ is the same. Since this is all the randomness is used for, this means the distribution generating $c'$ and $\hat{c}$ is equivalent. Thus, $(m, \mathsf{pk}, c, \mathsf{Rand}(c, \mathsf{pk}; U^*)) \equiv (m, \mathsf{pk}, c, \mathsf{Enc}(m, \mathsf{pk}; U^*))$.

Neutrality is actually implied by the fact that these two distributions are identical. Since there is no difference in the distributions of the re-encryption of $c$ and a fresh encryption of $m$, and decryption is always correct on a fresh encryption, decryption will still be correct on the re-randomized ciphertext. □

ElGamal public keys are already defined over an abelian group (multiplication in $\mathbb{Z}_p$), and the operation is efficient. For adding and removing layers, we define the functions AddLayer and DelLayer, both requiring the secret key and using Rand.

**function** $\mathsf{AddLayer}(c = (c_1, c_2), \mathsf{pk}_1, (\mathsf{pk}_2, \mathsf{sk}_2))$
  **return** $\mathsf{Rand}((c_1, c_1^{\mathsf{sk}_2} \cdot c_2), \mathsf{pk}_1 \cdot \mathsf{pk}_2)$
**end function**
**function** $\mathsf{DelLayer}(c = (c_1, c_2), \mathsf{pk}_1, (\mathsf{pk}_2, \mathsf{sk}_2))$
  **return** $\mathsf{Rand}((c_1, c_1^{-\mathsf{sk}_2} \cdot c_2), \mathsf{pk}_1)$
**end function**

After adding a layer with $\mathsf{sk}_2$ (corresponding to $\mathsf{pk}_2 = g^{\mathsf{sk}_2}$) to a ciphertext under $\mathsf{pk}_1$ (corresponding to $\mathsf{sk}_1$), our new public key is $g^{\mathsf{sk}_1+\mathsf{sk}_2} = \mathsf{pk}_1 \cdot \mathsf{pk}_2$. We will now show that AddLayer and DelLayer work as we expect. That is, adding a layer is indistinguishable from a fresh encryption under the combined public key, and deleting a layer is indistinguishable from a fresh encryption of the ciphertext with that key's component removed.

**Lemma 3.2** ([2])**.** *For any public key* $\mathbf{k}_1$ *and* $(\mathsf{pk}_2, \mathsf{sk}_2)$ *in ElGamal, and* $c = (c_1, c_2) = \mathsf{EG.Enc}(m, \mathsf{pk}_1)$*, the following are equivalent distributions:*

$$\{\mathsf{EG.AddLayer}(c, \mathsf{sk}_2)\} \equiv \{\mathsf{EG.Enc}(m, \mathbf{k}_1 \cdot \mathsf{pk}_2)\}$$

*and for* $c' = \mathsf{Enc}(m, \mathbf{k}_1 \cdot \mathsf{pk}_2)$*,*

$$\{\mathsf{EG.DelLayer}(c', \mathsf{sk}_2)\} \equiv \{\mathsf{EG.Enc}(m, \mathbf{k}_1)\}$$

*Proof.* For both equivalences, we will show that the intermediate input from AddLayer or DelLayer to Rand is a valid ciphertext under the combined or un-combined public key. First, we notice that $(c_1, c_2)$ is a ciphertext encrypting the message $m$ under the key $\mathbf{k}$ if and only if there exists an $r \in \mathbb{Z}_{p-1}$ such that $c_1 = g^r$ and $c_2 = \mathbf{k}^r \cdot m$. So, let $c = (g^r, \mathbf{k}_1^r \cdot m)$ and $c' = (c'_1, c'_2) = (g^r, (\mathbf{k}_1 \cdot \mathsf{pk}_2)^r \cdot m)$.

For AddLayer, we return $\mathsf{Rand}\left((c_1, c_1^{\mathsf{sk}_2} \cdot c_2)\right)$. The input to Rand, when expanded out, is

$$(c_1, c_1^{\mathsf{sk}_2} \cdot c_2) = (g^r, g^{r \cdot \mathsf{sk}_2} \cdot g^{\mathsf{sk}_1 \cdot r} \cdot m) = (g^r, (\mathbf{k}_1 \cdot \mathsf{pk}_2)^r \cdot m).$$

This is a valid encryption of $m$ under the public key $\mathbf{k}_1 \cdot \mathsf{pk}_2$. Now, from lemma 3.1, we know that the image of Rand is a fresh ciphertext under the public key $\mathbf{k}_1 \cdot \mathsf{pk}_2$.

We will show that DelLayer works in a similar way. It calls Rand with

$$(c'_1, c_1'^{\mathsf{sk}} \cdot c'_2) = (g^r, g^{-\mathsf{sk}_2 \cdot r} g^{\mathsf{sk}_1 \cdot r + \mathsf{sk}_2 \cdot r} \cdot m) = (g^r, \mathbf{k}_1^r \cdot m).$$

This is a valid ciphertext encrypting $m$ under public key $\mathbf{k}_1$. Thus, the ciphertext Rand returns is equivalent to a ciphertext generated by a fresh encryption of $m$ under that public key. □

This corollary is just saying that randomly choosing a new key-pair and adding a layer to a ciphertext is equivalent to generating a new public key and freshly encrypting the message under that key. Thus the requirement for "public-key re-randomization" holds.

**Theorem 3.3** ([2])**.** *ElGamal is a PKCR encryption scheme.*

*Proof.* ElGamal is already known to be a semantically secure public key cryptosystem. We are just adding Rand, AddLayer, and DelLayer. Lemmas 3.1 and 3.2 show that these functions work as they should in a PKCR-encryption scheme. □

### 3.1.3 Instantiation of PKCR-enc under QR

We will be using an instantiation of Cocks' Identity-Based Encryption scheme (IBE) [11]. One of the main issues with using a scheme like this is requiring a shared RSA modulus $N = pq$. To instantiate the protocol with QR, one would need a trusted-third-party to give all participating parties the RSA modulus (thus trusting the third party with the master secret). This could mean the parties directly interact with this or use a NIST codebook. In general, it is much harder for parties to agree upon a secure RSA modulus than, as in ElGamal, a DDH-secure prime. So, the primary use-case for PKCR with QR is for when an RSA modulus is being used already for another functionality in which all parties are also involved in. For example, if Cocks' IBE is already being used, parties can then continue using their public and secret keys.

The scheme was shown to be homomorphic by Joye [24]. Joye's work was extended to show that the Cocks' scheme can be used for two-way proxy re-encryption by LaVigne [27]. Using the same techniques from LaVigne's work, we can show that this IBE can be compiled into a PKCR encryption scheme. Because these techniques are much more involved than with ElGamal, we have moved the proofs to the appendix.

Instead of using Cock's IBE as an identity-based scheme, we will be using the simpler public-key version, which is all we need for PKCR-encryption — it is straightforward to convert the algorithms from this version to the full IBE. Here is a quick review of the public-key scheme. For details on the QR assumption and the Jacobi symbol, see appendix A.1.

- $\mathsf{QR.Gen}(1^\kappa)$ choose an $\kappa$-secure RSA modulus $N = pq$, $r \xleftarrow{\$} \mathbb{Z}_N^*$, and $R \leftarrow r^2 \pmod N$. Output the public-secret key pair $\mathsf{pk} = (N, R)$ and $\mathsf{sk} = r$.
- $\mathsf{QR.Enc}(m \in \{0,1\}, \mathsf{pk})$. Sample $t \xleftarrow{\$} \mathbb{Z}_N$ until $\left(\frac{t}{N}\right) = (-1)^m$ (where $\left(\frac{t}{N}\right)$ denotes the Jacobi symbol). Output the ciphertext $t + Rt^{-1}$.

- QR.Dec($c$, sk). Compute $\left(\frac{2r+c}{N}\right) = (-1)^{m'}$. Output $m'$.

Again, just as with ElGamal, we need to fix some parameters before generating public keys. In this case, we need to fix an RSA modulus $N$. From there, public keys are just squares in $\mathbb{Z}_N^*$, and just as public keys in ElGamal, the group operation is just multiplication mod $N$: note that a square multiplied by a square in $\mathbb{Z}_N^*$ is still a square in $\mathbb{Z}_N^*$, which represents a combined public key with the corresponding secret key being the square root (which is also the product of the two original roots).

Unfortunately, using quadratic residues is more technical than using DDH, and so full descriptions and proofs of QR.Rand, QR.AddLayer, and QR.DelLayer are in Appendix A.

**Theorem 3.4.** *Cocks' PKE is a PKCR encryption scheme.*

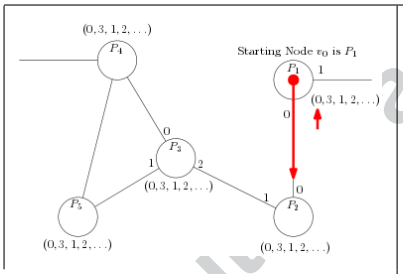
*Proof.* Cocks' PKE is already a semantically-secure public key scheme. Lemma A.5 shows that QR.Rand satisfies the randomization and neutrality requirements for a randomizing function. Lemma A.6 shows that QR.AddLayer and QR.DelLayer appropriately add and remove layers of public keys according to the definition of PKCR encryption. Therefore, the whole scheme is a PKCR scheme. □

## 3.2 Random Walks and Other Graph Exploration Sequences

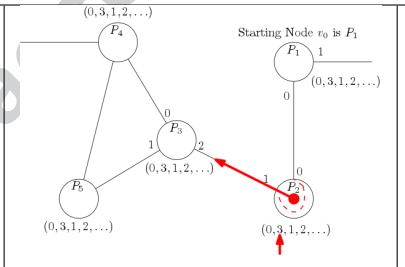
A key element in designing our algorithm is an *exploration sequence*. Informally, it is a sequence of edges that, when given a node to start on, traverses the entire graph. Before going into the formal definition, let us first define a few graph terms.

**Definition 3.5.** A *walk* (of length $T$) on an undirected graph $G = (V, E)$, where $|V| = n$ and $|E| = m$ is a sequence of vertices $(v_0, \ldots, v_T)$ such that for all $i \in [k]$, $(v_{i-1}, v_i)$ is an edge. A walk can visit the same vertex multiple times, or take the same edge.

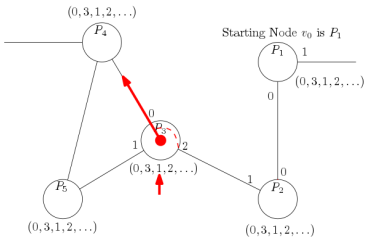
Next we will discuss how to induce a walk on a graph with just a sequence of integers, shown also in figure 3.1.




An exploration sequence starting at the node for party $P_1$. Because the sequence starts with 0, the first step is to take the edge labeled with 0 to node $P_2$.




The second element in the sequence is 3. Since the walk entered node $P_2$ through the edge it labeled 0, it will leave the edge labeled $1 \equiv 0 + 3 \mod \deg(P_2)$.




The next element in the sequence is 1, and since the walk entered node $P_3$ through the edge it labeled 2, it will leave through the edge labeled $0 \equiv 1 + 2 \mod \deg(P_3)$.

Figure 3.1: A figure demonstrating how to induce a walk on a graph when given an exploration sequence and starting node.

**Definition 3.6** ([26])**.** Let $G$ be a graph and every node $v \in V$ label its edges 0 to $\deg(v) - 1$ (as in figure 3.1). A sequence of integers $(\tau_1, \ldots, \tau_T)$ and starting node $v_0$ *induces* a walk on a graph $G$ in the following way: the walk starts on $v_0$ and the first step it takes is the edge labeled $\tau_1 \mod \deg(v_0)$ from $v_0$ to $v_1$; at step $i$, if the walk entered vertex $v_i$ from edge $j \in \{0, \ldots, \deg(v_i) - 1\}$, then it leaves $v_i$ to go to $v_{i+1}$ on the edge labeled $j + \tau_i \mod \deg(v_i)$. The sequence $(\tau_1, \ldots, \tau_T)$ is called an *exploration sequence*.

(Exploration) Sequence and walk will often be used interchangeably because each node will have a labeling for their edges so a sequence induces a walk and vice-versa. Next, we define the sequences we want more formally.

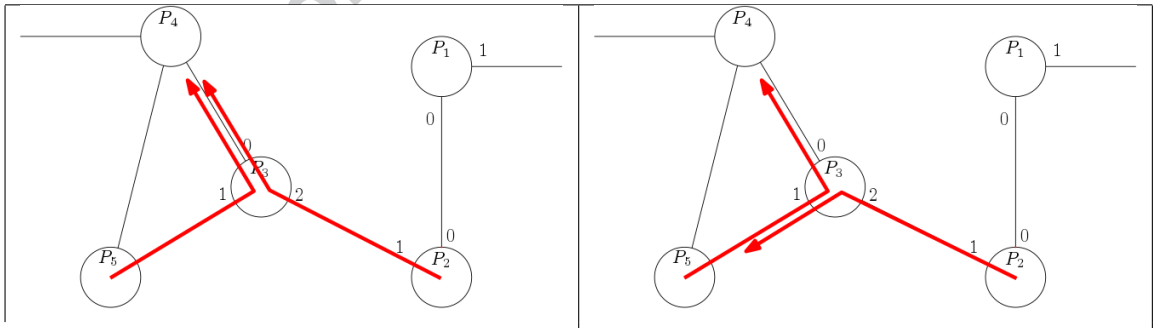
If $G$ is a $d$-regular graph, then $(\tau_1, \ldots, \tau_T) \in \{0, \ldots, d-1\}$. If such a sequence and starting edge ends up covering the entire graph, then it is an exploration sequence for $G$. If the sequence will cover all graphs of $n$ nodes from *any* starting edge, then we call it a universal exploration sequence. We will more explicitely define these below.

**Definition 3.7.** A *universal 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 consider a "universal" exploration sequence with errors as well. In this model, we consider the $\tau_i$'s are generated by some random process, and so there could be some probability that the walk fails to visit every node.

**Definition 3.8.** An *$\delta$-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 node $(v_0)$ so that the resulting walk $(v_0, v_1, \ldots, v_T)$ covers every $n$-node graph with max degree $d$ with probability at least $1 - \delta$ over the randomness used to generate the sequence.[3]

We will be using these sequences heavily in our protocol for topology-hiding OR: an encrypted message will make its way through the graph following an exploration sequence. However, one exploration sequence will end up meaning only one party gets output. To get correctness, we will need to run at least one of these walks per node. But, if we just run any series of exploration sequences, the way they "interfere" could reveal something about the topology of the graph, demonstrated in figure 3.2. So, to make our protocol topology-hiding, we want to have one walk per direction on each edge (so a total of $2 \cdot |\text{edges}|$ walks). For example, imagine that each undirected edge is actually two pipes from each of the nodes: one pipe going from the first to the second and the other pipe from the second to the first; there can only be one walk occupying one direction at each step. See also figure 3.3 for an illustration of what a single step of all of these walks running at once looks like. The reason for requiring this property is that it makes it easier to describe and analyze our protocols—the topology of the graph cannot interfere with the walks when define them to have this property.




An example of two exploration sequences interfering with each other. Notice because these two sequences are taking the same edge, $P_4$ learns that $P_3$ has at least two neighbors.

Instead, we want to make sure our sequences do not interfere, so there is at most one walk per edge, per direction at each step.

Figure 3.2: A figure demonstrating the difference between interfering sequences and non-interfering sequences.

[3]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.

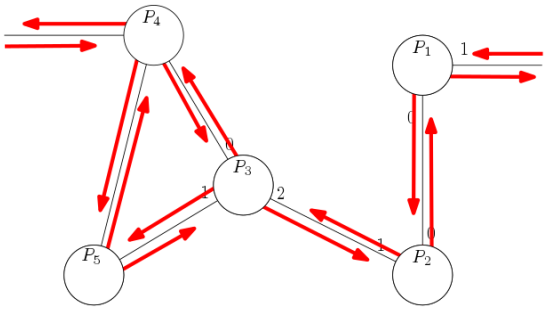



Figure 3.3: What any single step of a non-interfering full collection of covering sequences looks like: one walk traverses each direction of each edge.

It will be helpful to define this collection modularly. First, we will start with what a "full" collection is—in essence, it is a collection of sequences meant to fill all of the pipes in a graph (at least in the first step).

**Definition 3.9.** A *full collection* of sequences for graphs on $n$ nodes with $m$ edges is a collection of $2m$ sequences that each start on a different edge or edge-direction.

Now we can define what it means for this collection not to interfere with itself.

**Definition 3.10.** A *non-interfering* full collection of exploration sequences for graphs on $n$ nodes with $m$ edges is group of $2m$ exploration sequences such that if they run simultaneously, 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 all of the walks; if a walk enters node $v$ from edge $i$ at round $t$, it leaves that node from edge $\pi_{v,t}(i)$. So, we define the following function taking a node $v$, a time $t$, and outputting a permutation $\pi_{v,t} \in \mathcal{S}_{d_v}$ where $d_v$ is the degree of $v$:

$$\mathsf{Seq} : (v, t) \mapsto \pi_{v,t}$$

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

**Definition 3.11** ([2])**.** 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.[4]

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 (exploration sequences) also fit this description. We will analyze and compare how well these objects do in section 5.4.

**Remark.** If we have can compute a full collection of non-interfering information-local exploration sequences given the exact number of nodes $n$, then we also can compute full collection of non-interfering information-local exploration sequences for graphs on *at most* $n$ nodes.

We can do this simply by computing the exploration sequence for every $i = 1$ to $n$ and concatenating those sequences together. The length of the resulting sequence will still be polynomial in $n$ and require only local computation. There may be more efficient ways to do this depending on how one constructs these exploration sequence (whether they are random or not, for example).

[4]The definition proposed by [2] generalizes this one with $k$-information-local functions. We only care about 0-information-local functions for this work.

### 3.2.1 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 negl($\kappa$)-exploration sequences.

**Definition 3.12.** A *random walk* starting at node $v_0$ is a walk $(v_0, \ldots, v_T)$ such that each subsequent step is chosen independently at random. That is at step $i$, the walk is at node $v_i$, and step $i+1$ is at $v_{i+1}$, a uniformly-random chosen neighbor of $v_i$.

Correlated random walks are simply random walks that do not interfere with each other. That is, if two walks enter the same node, they must leave that node on different edges. This will ensure there is no more than one walk per direction per edge.

**Definition 3.13.** In this work, a set of correlated random walks will be a set of random walks $S = \{w_1, \ldots, w_k\}$, where step $t$ of each walk is determined randomly, conditioned on the fact that it does not interfere with any other walk. That is, if at step $i$, a subset of walks $S' \subset S$ are all at vertex $v_i$, then first, $|S'| \leq \deg(v_i)$ because no walks are allowed to interfere, and second, we choose $\binom{\deg(v_i)}{|S'|}$ distinct edges out from $v_i$ and then randomly choose which walks take which edges.

By theorem 5.4, 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)[29].

**Definition 3.14** (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 3.15** (Cover time bound)**.** *The cover time of any connected, undirected graph $G = (u, v)$ is bounded above by $4nm \leq 4n^3$.*

**Corollary 3.16.** *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 3.15 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 $\tau$ 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 3.17.** *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 3.16, we also know that each walk has probability $2^{-\kappa} = \text{negl}(\kappa)$ of covering the entire graph. The lemma follows immediately. □

### 3.2.2 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 5.4.2 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 [26].

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 3.18** ([34])**.** *There exists a polynomial length exploration sequence for all graphs on n nodes which can be computed in polynomial time given only n.*

So, what we have is a sequence that every node in a graph can compute locally (recall that an exploration sequence explores *all* graphs on $n$ nodes, and so computation does not require knowledge about the graph), and in polynomial time. We just need to prove that if we run these UESs simultaneously, they will not interfere.

**Lemma 3.19.** *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 3.18, 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 will prove this by induction on the number of steps in the walks. In the statement of this lemma, since we have a full collection of these sequences, each sequence starts at a different edge and direction. This means that at the first step of the algorithm, no sequences will interfere. 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 $t$: $\tau_t$. 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. □

## 4 A Stronger Simulation-based Definition of Topology-Hiding

Here we adapt the simulation-based definition of topology-hiding from [30] 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 [30]). It is important to note that [30], [23], and [2] all work within our stronger model; our contribution here is defining it and proving security within it. Our definition, similar to [30], will 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, [30] define the $\mathcal{F}_{\text{graph}}$-hybrid model, which employs a special "graph party," $P_{\text{graph}}$. Figure 4.1 shows $\mathcal{F}_{\text{graph}}$'s functionality: at the start of the functionality, $\mathcal{F}_{\text{graph}}$

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

**Participants/Notation:**

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

**Initialization Phase:**

**Inputs:** $\mathcal{F}_{\text{graph}}$ waits to receive the graph $G = (V, E)$ from $P_{\text{graph}}$, 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 $(\ell, m)$ 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 $(\ell, m)$ to $P_w$, where $f(v, w) = \ell$, indicating that the neighbor on edge $\ell$ sent the message $m$ to $P_w$.

Figure 4.1: The functionality $\mathcal{F}_{\text{graph}}$ with edge labels. Note that since the graph is undirected, $(u, v) = (v, u) \in E$ and so $f(u, v) = f(v, u)$.

receives the network graph from $P_{\text{graph}}$, 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 $P_{\text{graph}}$ 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 $P_{\text{graph}}$ (as in [30]), but—unlike [30]—$\mathcal{F}_{\text{Graph}}$ does not output the neighbors to each party. Instead, $\mathcal{F}_{\text{graph}}$ reveals edge-labels. These 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 with another node, but not enough to be able to tell if two nodes share a neighbor. We capture this leakage 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 $(\mathcal{F}_{\text{graphInfo}} \| \mathcal{F})$. For more details on this definition and exact model, see Appendix B.1.

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

**Definition 4.1.** We say that a protocol $\Pi$ 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.

## 4.1 Differences of this Model: Neighbors of Neighbors

In the first model, proposed by [30], $\mathcal{F}_{\text{graphInfo}}$ 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.

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

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

**Inputs:** Each party $P_i$ receives an input bit $b_i \in \{0, 1\}$.

**Outputs:** All parties $P_1, \ldots, P_n$ receive output $b = \text{OR}(b_1, \ldots, b_n)$.

Figure 4.2: The functionality $\mathcal{F}_{OR}$.

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

**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 4.3: The functionality $\mathcal{F}_{\text{Broadcast}}$.



### 4.2 OR functionality: $\mathcal{F}_{OR}$

In accordance with this definition, we need to define an ideal functionality of OR, denoted $\mathcal{F}_{OR}$, shown in figure 4.2. We will prove that a simulator only with knowledge of the output of $\mathcal{F}_{OR}$ and of the local topology of the adversarially chosen parties $Q$ can produce a transcript to $Q$ indistinguishable from running the protocol.

Our protocol for OR will provide the backbone of our Broadcast functionality, which is why we define it here.

### 4.3 Broadcast functionality: $\mathcal{F}_{\text{Broadcast}}$

The functionality $\mathcal{F}_{\text{Broadcast}}$ is included here in figure 4.3;as it will be the base protocol for which we will be able to build all other functionalities. Just as with $\mathcal{F}_{OR}$, we will be able to produce a simulator that can produce a transcript for the communication of adversarially corrupted parties $Q$ indistinguishable from a real-world running of our protocol, given only knowledge of the broadcasted bit and the local topology of $Q$.

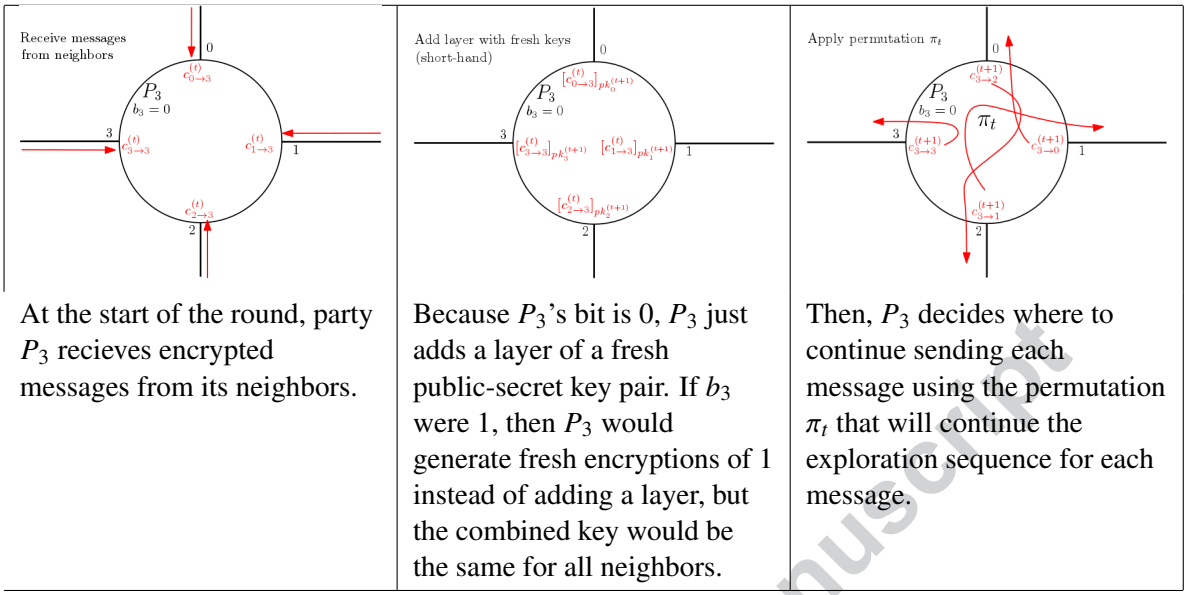
## 5 Topology Hiding Protocols for General Graphs

In this section, we describe two protocols: OR and Broadcast, proving they are complete and secure. We will see that Broadcast is just a special case of our OR. Once we have broadcast, we will be able to use it to realize any efficient function in a topology-hiding way.

### 5.1 Topology-Hiding OR Protocol

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 3.8) 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 $\pi_t^{-1}(j)$.




At the start of the round, party $P_3$ recieves encrypted messages from its neighbors.

Because $P_3$'s bit is 0, $P_3$ just adds a layer of a fresh public-secret key pair. If $b_3$ were 1, then $P_3$ would generate fresh encryptions of 1 instead of adding a layer, but the combined key would be the same for all neighbors.

Then, $P_3$ decides where to continue sending each message using the permutation $\pi_t$ that will continue the exploration sequence for each message.

Figure 5.1: A figure illustrating the view of one node's communication during the protocol in the aggregate phase at step $t$.

**Proof of simulation security** Recall that a protocol $\Pi$ is a *topology-hiding OR* protocol if it is a topology-hiding realization of $\mathcal{F}_{OR}$. In this section, we will show that protocol 1 is a topology-hiding OR protocol, satisfying definition 4.1. We will break the proof up into two parts. First lemma 5.1 will show that our protocol outputs the correct bit with all but negligible probability. We will then use lemma 5.1 to prove the full security in lemma 5.2.

**Lemma 5.1.** *Given a full collection of non-interfering, information-local, $\delta$-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 $\kappa$.*

*Proof.* Consider one sequence, or walk, in the collection of exploration sequences. We will prove that by the end of our protocol, the party that initiated that sequence will get the decrypted output bit. That is, we will show that every node along the sequence OR's its bit, and then 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 $i$th node in the sequence. For the sake of notation, $\mathsf{pk}_i$ will denote the public key generated by node $u_i$ at step $i+1$ for node $u_{i+1}$ (so $\mathsf{pk}_i = \mathsf{pk}^{(i+1)}_{u_i \to u_{i+1}}$), and $\mathbf{k}_i$ will be the aggregate key-product at step $i$ (so $\mathbf{k}_i = \mathsf{pk}_0 \circledast \ldots \circledast \mathsf{pk}_i$).

- On the first step, $u_0$ encrypts $b_0$ with $\mathsf{pk}_0$ into $c_1$ and sends it and public key $\mathsf{pk}_0$ 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 $\mathbf{k}_{i-1} = \mathsf{pk}_0 \circledast \mathsf{pk}_1 \circledast \cdots \circledast \mathsf{pk}_{i-1}$, also sent to $u_i$. $u_i$ computes the new public key $\mathsf{pk}_0 \circledast \cdots \circledast \mathsf{pk}_i = \mathbf{k}_i$, adding its own public key to the product.

  Now, $u_i$ computes the new ciphertext. If its input bit, $b_i$, is 0, then OR'ing it to the ciphertext will not change the value. So, $u_i$ just adds its own public key to the encryption of $c_i$, computing

**Protocol 1** Topology-hiding OR 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** $\mathsf{OR}(n, \kappa, d_i, b_i)$
  // 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 $(\mathsf{pk}^{(t)}_{i\to d}, \mathsf{sk}^{(t)}_{i\to d}) \leftarrow \mathsf{KeyGen}(1^\kappa)$.
  Generate $T-1$ random permutations on $d_i$ elements $\{\pi_1, \cdots, \pi_{T-1}\}$. Let $\pi_T$ be the identity permutation.
  // **Aggregate phase**
  For all $d \in [d_i]$, send to neighbor $d$ a fresh encryption $\mathsf{Enc}(b_i, \mathsf{pk}^{(1)}_{i\to d})$ and the public key $\mathsf{pk}^{(1)}_{i\to d}$.
  **for** $t = 1$ to $T-1$ **do**
    **for** Neighbors $d \in [d_i]$ **do**
      Wait to receive ciphertext $c^{(t)}_{d\to i}$ and public key $\mathbf{k}^{(t)}_{d\to i}$.
      Let $d' \leftarrow \pi_t(d)$.
      Compute $\mathbf{k}^{(t+1)}_{i\to d'} = \mathbf{k}^{(t)}_{d\to i} \circledast \mathsf{pk}^{(t+1)}_{i\to d'}$.
      **if** $b_i = 1$ **then**
        Compute $\hat{c}^{(t+1)}_{i\to d} \leftarrow [1]_{\mathbf{k}^{(t+1)}_{i\to d'}}$ // encryption of 1 under key $\mathbf{k}^{(t+1)}_{i\to d'}$
      **else** // $b_i = 0$
        Compute $\hat{c}^{(t+1)}_{i\to d} \leftarrow \mathsf{AddLayer}\left(c^{(t)}_{d\to i}, \mathbf{k}^{(t)}_{d\to i}, \mathsf{sk}^{(t+1)}_{i\to d'}\right)$.
      **end if**
      Send $c^{(t+1)}_{i\to d'}$ and $\mathbf{k}^{(t+1)}_{i\to d'}$ to neighbor $d'$.
    **end for**
  **end for**
  Wait to receive $c^{(T)}_{d\to i}$ and $\mathbf{k}^{(T)}_{d\to i}$ from each neighbor $d \in [d_i]$.
  **if** $b_i = 1$ **then**
    compute $e^{(T)}_{d\to i} \leftarrow [1]_{\mathbf{k}^{(T)}_{d\to i}}$.
  **else**
    Let $e^{(T)}_{d\to i} = c^{(T)}_{d\to i}$.
  **end if**
  // **Decrypt phase**
  **for** $t = T$ to $1$ **do**
    For each $d \in [d_i]$, send $e^{(t)}_{i\to d'}$ to $d' = \pi_t^{-1}(d)$. // Passing back
    **for** $d \in [d_i]$ **do**
      Wait to receive $e^{(t)}_{d\to i}$ and public key $\mathbf{k}^{(t)}_{d\to i}$ from neighbor $d$.
      Compute $d' \leftarrow \pi_t^{-1}(d)$.
      $e^{(t-1)}_{i\to d'} \leftarrow \mathsf{DelLayer}\left(e^{(t)}_{d\to i}, \mathbf{k}^{(t)}_{d\to i}, \mathsf{sk}^{(t)}_{i\to d'}\right)$ // If $t = 1$, DelLayer decrypts.
    **end for**
  **end for**
  // **Produce output bit**
  $b \leftarrow \bigvee_{d\in[d_i]} e^{(0)}_{i\to d}$.
  Output $b$.
**end procedure**

$c_{i+1}$ as $\mathsf{AddLayer}(c_i, \mathsf{pk}_i)$. However, if $b_i$ is 1, then OR'ing it to the ciphertext will ensure that the ciphertext is encrypting 1. So, $u_i$ just computes $c_{i+1}$ as fresh encryption of 1 under the combined public key $\mathbf{k}_i$.

In both cases, $c_{i+1}$ now encrypts $b_0 \vee \ldots \vee b_i$ under key $\mathbf{k}_i$.

- 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 $\mathsf{pk}_0 \circledast \cdots \circledast \mathsf{pk}_{T-1} = \mathbf{k}_{T-1}$. $u_T$ then OR's his own bit with the procedure described in the previous steps. If $b_T$ is 0, then the ciphertext $e_T = c_{T-1}$. If $b_T$ is 1, then $e_T = [1]_{\mathbf{k}_{T-1}}$. $u_T$ sends $e_T$, an encryption of $b_0 \vee \ldots \vee b_T$ under $\mathbf{k}_{T-1}$, 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 $\mathsf{pk}_1 \circledast \cdots \circledast \mathsf{pk}_i = \mathbf{k}_i$. $u_i$ deletes the key layer $\mathsf{pk}_i$ to get $\mathbf{k}_{i-1}$ and then using $\mathsf{DelLayer}$, removes that key from encrypting $e_i$ to get $e_{i-1}$. $u_i$ sends $e_{i-1}$ and $\mathbf{k}_{i-1}$ to $u_{i-1} = (\pi_i^{(u_i)})^{-1}(u_{i+1})$.

- Finally, node $u_0$ receives $e_0$ encrypted only under public key $\mathsf{pk}_0$ on step 1. $u_0$ deletes that layer $\mathsf{pk}_0$, 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

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

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

We will now use the completeness of our protocol to show topology-hiding security from definition 4.1.

**Theorem 5.2.** *If the underlying PKCR encryption scheme is CPA-secure and a full collection of non-interfering, information-local, $\delta$-Exploration Sequences for any $\delta = negl(\kappa)$ of length $T$, then protocol 1 realizes the functionality of $\mathcal{F}_{OR}$ 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 $\mathcal{F}_{OR}$. $\mathcal{F}_{OR}$ outputs bit $b_{out}$ and sends it to $\mathcal{S}$. Note $\mathcal{S}$ only requires knowledge of $Q$'s inputs and the output of $\mathcal{F}_{OR}$ for this step.

3. $\mathcal{S}$ gets the local neighborhood for each $P \in Q$: $\mathcal{S}$ knows how many neighbors each $P$ has, and if a neighbor is also in $Q$, $\mathcal{S}$ knows which party that is. $\mathcal{S}$ doesn't need to know anything else about the topology. $\mathcal{S}$ denotes $\mathcal{N}(P)$ the neighborhood of pseudonyms for each $P \in Q$. [5].

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

[5] Recall that from definition 4.1, $\mathcal{F}_{\text{graphInfo}}$ 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:

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

  At step $t = i$ in the for loop on line 8, $\mathcal{S}$ simulates $Q$ sending a message to $P$ by sending $([0]_{\mathsf{pk}^{(i)}_{Q\to P}}, \mathsf{pk}^{(i)}_{Q\to P})$. $\mathcal{S}$ receives the pair $(c^{(i)}_{P\to Q}, \mathbf{k}^{(i)}_{P\to Q})$ 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 $[b_{out}]_{\mathbf{k}^{(i)}_{Q\to P}}$ 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 using many hybrids. The hybrid structure is multidimensional, where each dimension represents a finer gradation. The order of these multi-dimensional hybrids, denoted by $h.(i,t)$, is first $t$ goes from 0 to $T$ or $2T$ (the number of rounds), then when $t$ gets to $T$ or $2T$, $i$ increases and $t$ goes back to 1; when both $i$ (representing the number of neighbors of each corrupt party) and $t$ reach their upper bounds, we increase the hybrid number $h$. For example, the sub-hybrids of 2 are ordered $2.(1,1)$, $2.(1,2), \ldots, 2.(1,T), 2.(2,1), \ldots, 2.(|M|,1), \ldots, 2.(|M|,T)$.

(a) Hybrid 0. $\mathcal{S}$ 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 1. $\mathcal{S}$ now does pseudonym replacement. Now, $\mathcal{S}$ simulates the real world, but separates the simulated parties $P \in \mathcal{N}(\mathcal{Q})$ to parties in $\mathcal{N}'(\mathcal{Q})$, defined as follows. Consider the multiset $M = \bigcup_{Q\in\mathcal{Q}} \mathcal{N}(Q)$. Now let $\mathcal{N}'(\mathcal{Q})$ be the set $\{P'_1, \ldots, P'_{|M|}\}$, where each $P'_i$ corresponds to the real party in $M$ with that label. That is, if $P$ is the neighbor of both $Q_1$ and $Q_2 \in \mathcal{Q}$, then $P$ gets separated into two parties in $\mathcal{N}'(\mathcal{Q})$. Notice that in the pseudonym model, all the adversary has access to is this set of parties $\mathcal{N}'(\mathcal{Q})$, and cannot tell whether two of its parties share a neighbor. This hybrid makes this explicit for the simulator. So, in this hybrid, messages that would have been sent from $P_i \in \mathcal{N}(\mathcal{Q})$ to $Q \in \mathcal{N}(\mathcal{Q})$ are instead sent by the corresponding simulated party $P'_{i'} \in \mathcal{N}'(\mathcal{Q})$. $\mathcal{S}$ chooses pseudonyms for each $P'_{i'}$ at random from $[n^2]$.

(c) Hybrid 2. This hybrid consists of many sub-hybrids: we will replace the real keys that corrupted parties, $\mathcal{Q}$, see with simulated, freshly-generated public keys, but we will need to do that one key, one round at a time. For $m \in [|M|]$, at every round $t \in [T]$, we have a separate sub-hybrid $2.(i,t)$.

At hybrid $2.(i,t)$, for every $i' < i$, the keys that party $P'_{i'}$ would have given to $Q \in \mathcal{Q}$ have been replaced with freshly generated public keys. For $P'_i$, at every round $t' \le t$, we have replaced the key given to some $Q \in \mathcal{Q}$ at round $t'$ with a freshly generated public key. Messages sent are sent under this fresh public key instead of with a layered public key.

(d) Hybrid 3. $\mathcal{S}$ now will send encryptions of 0 during the aggregate phase regardless of what the actual bit is. To be formal about this, we split this into $|M| \cdot T$ rounds. Hybrid $3.(i,t)$ has the following structure: for every $i' < i$, all messages sent from simulated party $P'_{i'}$ are (fresh) encryptions of 0 under a fresh public key, and for every $t' \le t$, messages sent from simulated party $P'_i$ are always (fresh) encryptions of 0 under a freshly generated public key.

(e) Hybrid 4. $\mathcal{S}$ now starts simulating the decrypt phase, replacing the unlayered encryptions of the OR'd bits with fresh encryptions of them. Note that this is still not quite the ideal functionality since the bits we have OR'd together are generated by the walk, not necessarily $b_{out}$. We again need to split this hybrid into $|M| \cdot T$ sub-hybrids. So, hybrid $4.(i,t)$ is as follows: for all parties $i' < i$, $\mathcal{S}$ has the simulated party $P'_{i'}$ send a fresh encryption (under

the appropriate public key) of the bit generated by the walk, and for $t' \leq t$, $\mathcal{S}$ has party $P'_{i'}$ send fresh encryptions of this bit instead of the unlayered ones.

(f) Hybrid 5. $\mathcal{S}$ finally simulates the ideal functionality at the during the decrypt phase, sending fresh encryptions of $b_{out}$, the output of $\mathcal{F}_{OR}$, under the simulated public keys. This is instead of simulating the sequences through the graph and ORing only specific bits together.

Notice that hybrid 5 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 0 and 1. This will be because these distributions are identical in the view of the adversary. In Hybrid 0, the adversary sees pseudonyms chosen independently of whether or not a party is the neighbor to multiple corrupt parties, and so the pseudonyms are distributed identically to Hybrid 1. The messages sent in Hybrid 0 and Hybrid 1 are also identically distributed; messages from a specific party that was split into two or more in $\mathcal{N}'(Q)$ are just forwarded to the corresponding simulated party.

(b) First, we claim no adversary can distinguish between Hybrid 1 and 2.(1, 1). The difference between these hybrids is distinguishing between AddLayer and computing a fresh encryption key. The difference between hybrid 1 and 2.(1, 1) is simply that the first message sent by party $P'_1$ is encrypted under a fresh key instead, but this is also how the protocol starts. In general, the difference between hybrids 2.$(i, T)$ and 2.$(i + 1, 1)$ is non-existant. Now, the difference between hybrid 2.$(i, t)$ and 2.$(i, t+1)$ is that we change from using a layered public key to using a fresh public key to encrypt our bit. This is indistinguishable to an adversary since we claim that the following two distributions are equivalent in a PKCR encryption scheme for any $\mathbf{k} \in \mathcal{PK}$:

$$\{\mathbf{k} \circledast \mathsf{pk}_{new} | (\mathsf{pk}_{new}, \mathsf{sk}_{new}) \leftarrow \mathsf{KeyGen}(\kappa)\} \equiv \{\mathsf{pk}_{new} | (\mathsf{pk}_{new}, \mathsf{sk}_{new}) \leftarrow \mathsf{KeyGen}(\kappa)\}$$

and finally since these two distributions are equivalent and a fresh encryption is equivalent to a layered encryption, no adversary can distinguish between a layered encryption with a layered key and a fresh encryption with a fresh key.

(c) 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. We will examine how each sub-hybrid is computationally indistinguishable from the next. First, consider the difference between hybrid 2.$(|M|, T)$ and 3.$(1, 1)$. The difference here is that on the first message, party $P'_1$ may be sending an encryption of 0 instead of 1. Similarly, the difference between any 3.$(i, t)$ and 3.$(i, t + 1)$ or 3.$(i, T)$ and 3.$(i + 1, 1)$ is that the contents of an encrypted message may be 0 instead of 1. In all of these cases, due to the semantic security of the encryption scheme, a computationally-bounded adversary cannot distinguish between any of these hybrids.

(d) Next, we will show that Hybrids 3 and 4 are indistinguishable. For each sequential hybrid pair, of the form 3.$(|M|, T)$ and 4.$(1, 1)$ or 4.$(i, t)$ and 4.$(i, t + 1)$ or 4.$(i, T)$ and 4.$(i + 1, 1)$, the only difference is that the simulator gives a fresh encryption of a single message during the protocol to a corrupted party, instead of removing a layer from the real ciphertext that

has been simulated on the entire graph. From the definition of PKCR encryption, we know that the distribution of ciphertexts after removing a layer is equivalent to that of a fresh encryption under the resulting public key. This implies that these pairs are each individually indistinguishable.

(e) For this last case, we will show that there should not exist a PPT adversary $\mathcal{A}$ that can distinguish between Hybrids 4.($|M|, T$) and 5.

For any sequence that covers a set of $S$ nodes, let $b_S = \bigvee_{u \in S} b_u$, and notice that $b_{out} = \bigvee_{i \in [n]} b_i$, the OR of all parties' bits. Lemma 5.1 states that with all but negligible probability $b_S = b_{out}$. So, the chance that in one of these walks, the real-world bit $b_S$ is different from $b_{out}$ is negligible, making all of these sub-hybrids statistically indistinguishable. In fact, by a union bound, the probability that any of these walks fails is at most $2n^2 \cdot \text{negl}(\kappa)$, which is still negligible.

□

## 5.2 Topology-Hiding Broadcast from Topology-Hiding OR

Broadcast can be seen as just a special case of OR: there is a source-node $s$ with a source bit $b_s$, and all other parties have 0 as their input bits. By the end of the protocal, every protocol will have the $OR(0, \ldots, 0, b_s, 0, \ldots, 0) = b_s$. In this section, we will formally prove that topology-hiding OR implies topology-hiding broadcast, and then get the corollary that we achieve topology-hiding broadcast from the previous subsection.

**Lemma 5.3.** *If protocol* $\Pi$ *realizes topology-hiding OR, then there exists a protocol* $\Pi'$ *that achieves topology-hiding broadcast. Moreover,* $\Pi$ *and* $\Pi'$ *have the same communication and round complexity.*

*Proof.* $\Pi'$ is actually a special case of $\Pi$. Parties all run protocol $\Pi$ with the following inputs: if party $P_i$ is not the source, then they run $\Pi$ with input bit $b_i = 0$; otherwise they are the source with broadcast bit $b_{source}$, and they will run $\Pi$ with input $b_i = b_{source}$. After $\Pi$ finishes, then by the correctness of $\Pi$ each party gets the output $b_{source} = OR(0, \ldots, 0, b_{source}, 0, \ldots 0)$.

$\Pi'$ is a topology-realization of $\mathcal{F}_{\text{Broadcast}}$. First, the output of $\Pi'$ matches the output for broadcast. Second (we now need to prove that there exists a simulator) because for any set of corrupted parties, there exists a simulator for $\Pi$ such that any PPT adversary cannot distinguish between interacting with the real network or with the simulator. We can just use this same simulator for $\Pi'$, using the inputs as determined above. □

Given we have a topology-hiding OR protocol and a method for using OR to compute a broadcast we now get one of our main results: topology-hiding broadcast for all graphs.

**Theorem 5.4** (Topology-hiding broadcast for all network topologies)**.** *Suppose there exists a PKCR encryption scheme and a full collection of information-local non-interfering* $\text{negl}(\kappa)$*-exploration sequences of length* $T = \text{poly}(\kappa)$*. Then, there exists a polynomial-time protocol* $\Pi_n$ *that is a topology-hiding broadcast over any network topology* $G$ *of at most* $n = \text{poly}(\kappa)$ *nodes. Moreover, this protocol takes* $2T$ *rounds.*

*Proof.* We will show that protocol 1 is the topology-hiding realization of $\mathcal{F}_{\text{Broadcast}}$ using the transformation from lemma 5.3. Recall that this is just having the broadcast node input the broadcast bit into the protocol, and all other parties inputting 0. Since we assume existence of a PKCR, we are able to run our protocol. The rest of this proof is simply combining the results of lemma 5.1 and theorem 5.2.

To show protocol 1 is complete, lemma 5.1 states that for our parameter $\kappa$, 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 $\kappa$.

To show our protocol is sound, theorem 5.2 states that for our input parameter $\kappa$, an adversary can distinguish a simulated transcript from a real transcript with probability negligible in $\kappa$. Therefore,

protocol 1 is sound against all PPT adversaries: they have only a negligible chance with respect to $\kappa$ of distinguishing the simulation versus a real instantiation of the protocol. □

### 5.3 Topology-Hiding Computation from PKCR

Now that we have shown protocol 1 can satisfy definition 4.1 for topology-hiding *broadcast* (by computing OR on the correct inputs) in lemma 5.2 and 5.3, we can formally state and prove the main theorem (that there is topology-hiding broadcast), and its corollary (that there is topology-hiding computation for all efficient functions).

As mentioned in section 1, we can use public-key cryptography to compile broadcast into multiparty computation in the UC model. Recall that for every functionality $\mathcal{F}$, we say that a protocol $\Pi$ is a *topology-hiding protocol for* $\mathcal{F}$ if it is a polynomial-time topology-hiding realization of $\mathcal{F}$.

**Theorem 5.5** (Topology-hiding computation for all network topologies)**.** *Suppose there exists a PKCR encryption scheme and a full collection of information-local non-interfering negl($\kappa$)-exploration sequences of length $T = poly(\kappa)$. Then for every polynomial-time functionality $\mathcal{F}$, there exists a protocol $\Pi_n$ that is a topology-hiding protocol for $\mathcal{F}$ over any network topology graph $G$ on at most $n$ nodes. Moreover, if there exists an MPC protocol running in $T'$ rounds, then there exists a topology-hiding protocol for $\mathcal{F}$, $\Pi_n$, that only takes $2T \cdot T' + 1$ rounds.*

*Proof sketch.* Because the UC model is very technical and requires more definitions and helper lemmas, the full proof and model is explained in Appendix section B. Here we will provide the backbone intuition.

Theorem 5.4 states that we can get topology-hiding broadcast in our model. So, given broadcast and a PKI (implied by our assumption that a PKCR encryption scheme exists), we explain how to simulate point-to-point channels between parties with the following protocol. In the first round, every party generates a public-secret key-pair $((\mathsf{pk}_i, \mathsf{sk}_i)$ for party $P_i)$, and broadcasts the public key to the network. Now, if party $P_i$ needs to send a message $m$ to party $P_j$, $P_i$ simply encrypts $m$ using $\mathsf{pk}_j$ and broadcasts this encrypted message. No other parties can read this message, and we can make sure to tag the encryption in such a way that $P_j$ knows it is for them. See section B.2 for more details.

Now, we know that there exists an efficient MPC protocol $\Pi_f$ for every efficiently-computable function $f$ that uses point-to-point channels [36, 16, 15]. We can compile $\Pi_f$ into $\Pi$, a topology-hiding realization of the functionality of $f$, in the following manner. First, all parties broadcast their public keys, keeping their secret keys private. Then, for every round of $\Pi_f$, for every message $P_i$ sends to $P_j$, $P_i$ encrypts it under $P_j$'s public key, and broadcasts it. Now, notice that these broadcasts can all happen simultaneously (as per the nature of the UC model — these protocols do not interfere with each other), so if $\Pi_f$ took $T'$ rounds and our broadcast takes $2T$ rounds, then $\Pi$ takes $1 + T' \cdot 2T = 2TT' + 1$ rounds. This accounts for 1 round of broadcasting public keys, and then all of the broadcasts. □

### 5.4 Complexity, Success Probability, and Optimizations for Broadcast and Computation

In this section, we will discuss the complexity of our protocol with a concrete realization of an exploration sequence: random walks. We will also discuss how to optimize the length of the walks if we restrict ourselves to certain kinds of graphs (for example, it requires much fewer steps to cover an expander than an arbitrary graph). We will also discuss our other method for exploring the graph: universal exploration sequences. Unfortunately, concretely nailing down the length of a locally (and efficiently) computed universal exploration sequence is difficult, but the guarantees will be stronger.

#### 5.4.1 Communication Complexity with Correlated Random Walks

There has fortunately already been research into how long it takes for a random walk to cover a graph, especially when it comes to analyzing different kinds of graphs. Here we consider expanders and regular graphs in addition to arbitrary graphs [29, 8, 25].

**Corollary 5.6.** *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 3.17 shows that with a collection of correlated random walks of length $\kappa \cdot 8n^3$, we get $2^{-\kappa}$-exploration sequences. By theorem 5.4, 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). □

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 5.7** (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 $n, \kappa$ is $\Theta(\kappa n^5)$ group elements.

**Corollary 5.8.** *On input* $n, \kappa$*, the communication complexity of protocol 1 is* $\Theta(\kappa n^5)$ *group elements.*

*Proof.* For a graph with at most $n$ nodes, $B = 4n^3$ is an upper bound on the cover time (see theorem 3.15), and $m = n^2$ is an upper bound on the number of edges. Assigning those $B, m$ in the bound from claim 5.7, 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.

For example, on expander graphs (arising for example in natural applications on random regular graphs), it is known that the cover time is $C_G = O(n \log n)$, much less than $O(n^3)$ [8]. 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.

### 5.4.2 The Benefits of Universal Exploration Sequences

There are two possible sources of error in the construction in corollary 5.6: first, the encryption scheme may have some small probability of error when decrypting, and second, some of the random walks could fail to cover the entire graph. We can actually get rid of the second source of error if instead of using random walks, we use UESs. So, if our encryption scheme is perfectly correct, we get a *perfectly-complete* topology-hiding broadcast protocol. As noted before, however, we do not have good bounds on the round-complexity for using a UES: while we know that UES's *exist* that are about the same length as random walks ($O(n^4 \log n)$ [3]), being able to construct one efficiently locally requires log space, which gives no bounds on the size of the output except polynomial [34]. If parties all recieved polynomial advice in the form of a UES for a graph of $n$ nodes, then parties could all just use the same UES of length $O(n^4 \log n)$ for the protocol, but advice is not something we considered in our model.

**Lemma 5.9.** *Assuming a perfectly-correct PKCR encryption scheme, there exists a polynomial-round topology-hiding broadcast for any graph G that always succeeds.*

*Proof.* First, a perfectly-correct PKCR encryption scheme is one such that after any combination of adding and deleting layers, it decryption is always correct. This means that as long as every walk passes every node in the graph at least once, the protocol will be correct and sound.

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. So, every walk results in the output bit, and hence every party gets the output bit, so protocol 1 is perfectly complete. □

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[26]. 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.[6] 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 5.10** ([34])**.** *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 [26].

**Theorem 5.11** ([26])**.** *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 5.12.** *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 5.10 in conjunction with theorem 5.11 $O(n^2)$ 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 subsequence 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 $O(n^2 T)$. □

[6] 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

| Assumption | PKCR Scheme | Exploration Sequence | Success Probability | Round Complexity | Bandwidth Per Round |
|---|---|---|---|---|---|
| DDH | ElGamal | UES's | 1 | $\text{poly}(\kappa)$ | $3 \log p$ |
| DDH | ElGamal | Random Walks | $1 - \text{negl}(\kappa)$ | $2 \cdot (\kappa \cdot 8n^3)$ | $3 \log p$ |
| QR | Cock's PKE | Random Walks | $1 - \text{negl}(\kappa)$ | $2 \cdot (\kappa \cdot 8n^3)$ | $2 \log N$ |
| QR | Cock's IBE | Random Walks | $1 - \text{negl}(\kappa)$ | $2 \cdot (\kappa \cdot 8n^3)$ | $3 \log N$ |
| LWE | Regev's PKE | Random Walks | $1 - \text{negl}(\kappa)$ | $2 \cdot (\kappa \cdot 8n^3)$ | $O(\kappa^2 \log p) + 2 \log p$ |

Table 2: A table comparing how the different assumptions can be used, their round complexity (given whether or not they use UES's or random walks), and bandwidth of messages sent each round. Since the ElGamal encryption scheme is the only one with zero-error (QR induces a small probability of error after re-randomizing ciphertexts, and LWE has error built-in), we do not consider the other two assumptions with UES's. Note that in every case, we need to send both the public key and the encrypted message.

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 $\kappa$, unlike the random walk construction. Therefore, it is more efficient to use UESs in this protocol if $n$ is small compared to $\kappa$. 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.

### 5.5 Complexity of Broadcast with Concrete Assumptions

Cover time directly correlates to round-complexity, while the assumption (DDH, QR, or LWE) correlates with total communication complexity. In this subsection we will analyze the communication complexity of our broadcast protocol assuming DDH, QR, or LWE, and compare them. We also note that DDH implies an error-free PKCR. From section 5.4.2, we know that this implies our broadcast protocol is perfectly correct under those assumptions.

Getting PKCR from LWE was recently done by [28]. They showed that Regev's PKE [33] was also a PKCR, allowing for adding and deleting layers while preserving the integrity of the ciphertext. Assuming LWE is quantum-secure, this result means get topology-hiding computation against quantum adversaries. The downside to using this scheme is that it is less efficient both in terms of computation and space: Regev LWE ciphertexts are much larger than ElGamal or Cock's ciphertexts.

**Corollary 5.13.** *Suppose one of DDH, QR,* or *LWE is true, and there is a full collection of information-local non-interfering negl($k$)-exploration sequences of length $T = \text{poly}(\kappa)$. Then, there exists a polynomial-time protocol $\Pi_n$ that is a topology-hiding broadcast over any network topology $G$ of at most $n = \text{poly}(\kappa)$ nodes. Moreover, this protocol takes $2T$ rounds.*
*Moreover, if DDH is true, then there exists a perfectly-correct polynomial-time protocol $\Pi'_n$ that takes $\text{poly}(\kappa)$ rounds.*

*Proof.* First, consider QR and LWE. Theorem 3.4 showed that given $N$ as a security parameter, Cock's PKE and IBE are both PKCR encryption schemes. Then, by a theorem from [28], we get that LWE is also a PKCR encryption scheme. Therefore, if either QR or LWE holds, theorem 5.5 states that we get topology-hiding computation for any functionality.

Theorem 3.3 shows that ElGamal is a PKCR encryption scheme. We also know that ElGamal is error-free, regardless of how many times Rand was used on a ciphertext, or the starting encryption randomness. With corollary 5.9, we realize a perfectly-correct topology-hiding broadcast, which in turn compiles into a perfectly-correct topology-hiding protocol for any polynomial-time functionality by theorem 5.5. □

| Work | Graphs | Round Complexity | Communication Complexity |
|---|---|---|---|
| Moran et. al. '15 [30] | Diameter $D$ | $2^{O(D)} \cdot \text{poly}(\kappa)$ | $2^{O(D)} \cdot \text{poly}(\kappa)$ |
| Hirt et. al. '16 [23] | Diameter $D$ maximum-degree $d$ | $5D$ | $O((d+1)^D n\kappa)$ |
| Akavia-Moran '17 [2] | Cycles, trees, low-circumference | $2n$ | $2n^2 \cdot \text{poly}(\kappa)$ |
| This work (random walks) | All | $16n^3\kappa$ | $16n^5\kappa \cdot \text{poly}(\kappa)$ |

Table 3: A table directly comparing the communication and round complexity of our broadcast protocol with that of [30], [23], and [2]. Note that [30], [23], and this work all yield protocols for general graphs, but [2] only applies to cycles and trees.
The first work, [30], employs a generic MPC protocol, and we were unabel to compute estimates on the round or communication complexity. We do know that the round complexity increases exponentially with the diameter.
The last term in the communication complexity of [2] and this work is the size of a single ciphertext.

## 6 Conclusion and Future Work

This work showed that topology-hiding computation is feasible for *every* network topology (in the computational setting, assuming DDH or QR), 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 [30, 23, 2]. As noted in table 3, our round and communication complexity is better than all previous works for general graphs (e.g. when diameter and max-degree both must be treated as the worst case $n$), though the use of random walks means it is less efficient for graphs with well defined structures (e.g. cycles and trees). The work by Hirt et. al. is more efficient than our protocol in terms of round complexity, but has a problem when it comes to communication complexity. Only when we are okay leaking an upper bound on the maximum degree of the communication graph *and* this upper bound is $O(\log(n))$ is the communication complexity close to being as efficient as our protocol.

All of these results were building towards a feasibility result against the weakest possible adversary. This leaves completely open the feasibility question against a malicious or adaptive adversary.

Although there are impossibility results for even very weak malicious adversaries (fail-stop) [30], one can consider models in which we are able to restrict the amount of information the adversary learns. To this end, since the publication of this result, there have been a few works extending these results. First, by assuming secure hardware, [4] was able to get topology-hiding computation on arbitrary graphs against fail-stop adversaries, leaking at most a single bit of information. Soon after, [28] was able to get the same result but only requiring one of the standard cryptographic assumptions: DDH, QR, or LWE.

Moreover, the impossibility result hinges on the fact that the adversary is able to disconnect the network. So, if we limit the adversary so that it cannot disconnect the graph, there is hope that we can get some results (say a $t$-connected graph and the adversary can abort/control at most $t$ nodes).

There is also the question of whether or not topology-hiding can be more efficient using standard assumptions. Given hardware, as in [4], round complexity can be optimal (based on diameter) against passive adversaries. However, for standard assumptions, we seem to be stuck with polynomial rounds in the number of parties and security parameter: quadratic or more. There may be many other strategies we have not discovered that are more efficient, or perhaps there are lower bounds implying stronger assumptions are necessary for fewer rounds.

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. The impossibility results do not rule out this model. Perhaps we can even adjust this relatively simple protocol to work for these kinds of graphs.

Topology hiding computation is still a relatively unexplored subject in cryptography, having (in the computational setting) its first feasibility result in 2015 [30]. It will be exciting to see what else can be proved in this model.

## References


[1] A. Akavia, R. LaVigne, and T. Moran. Topology-hiding computation on all graphs. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 447–467, 2017.

[2] A. Akavia and T. Moran. Topology-hiding computation beyond logarithmic diameter. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, pages 609–637, 2017.

[3] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovasz, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, SFCS '79, pages 218–223, Washington, DC, USA, 1979. IEEE Computer Society.

[4] M. Ball, E. Boyle, T. Malkin, and T. Moran. Exploring the boundaries of topology-hiding computation. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, pages 294–325, 2018.

[5] J. Balogh, B. Bollobs, M. Krivelevich, T. Mller, and M. Walters. Hamilton cycles in random geometric graphs. *The Annals of Applied Probability*, 21(3):1053–1072, 2011.

[6] A. Beimel, A. Gabizon, Y. Ishai, E. Kushilevitz, S. Meldgaard, and A. Paskin-Cherniavsky. Non-interactive secure multiparty computation. In J. A. Garay and R. 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.

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

[8] 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.

[9] N. Chandran, W. Chongchitmate, J. A. Garay, S. Goldwasser, R. Ostrovsky, and V. 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.

[10] M. Clear, A. Hughes, and H. Tewari. *Homomorphic Encryption with Access Policies: Characterization and New Constructions*, pages 61–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[11] C. Cocks. An identity based encryption scheme based on quadratic residues. In *Proceedings of the 8th IMA International Conference on Cryptography and Coding*, pages 360–363, London, UK, UK, 2001. Springer-Verlag.

[12] D. Estrin, R. Govindan, J. Heidemann, and S. 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.

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

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

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

[16] 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.

[17] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F. Liu, A. Sahai, E. Shi, and H. Zhou. Multi-input functional encryption. In P. Q. Nguyen and E. 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.

[18] S. D. Gordon, T. Malkin, M. Rosulek, and H. Wee. Multi-party computation of polynomials and branching programs without simultaneous interaction. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 575–591, 2013.

[19] S. Halevi, Y. Ishai, A. Jain, E. Kushilevitz, and T. 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.

[20] S. Halevi, Y. Lindell, and B. Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Rogaway [35], pages 132–150.

[21] S. Halevi, Y. Lindell, and B. Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Rogaway [35], pages 132–150.

[22] M. Hinkelmann and A. 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).

[23] M. Hirt, U. Maurer, D. Tschudi, and V. 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.

[24] M. Joye. Identity-based cryptosystems and quadratic residuosity. In *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, pages 225–254, 2016.

[25] J. D. Kahn, N. Linial, N. Nisan, and M. E. Saks. On the cover time of random walks on graphs. *Journal of Theoretical Probability*, 2(1):121–128, 1989.

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

[27] R. LaVigne. Simple homomorphisms of cocks IBE and applications. *IACR Cryptology ePrint Archive*, 2016:1150, 2016.

[28] R. LaVigne, C. L. Zhang, U. Maurer, T. Moran, M. Mularczyk, and D. Tschudi. Topology-hiding computation beyond semi-honest adversaries. In *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part II*, pages 3–35, 2018.

[29] M. Mitzenmacher and E. Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[30] T. Moran, I. Orlov, and S. Richelson. Topology-hiding computation. In Y. Dodis and J. B. Nielsen, editors, *TCC 2015*, volume 9014 of *Lecture Notes in Computer Science*, pages 169–198. Springer, 2015.

[31] M. Penrose. *Random geometric graphs*. Number 5. Oxford University Press, 2003.

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

[33] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93, 2005.

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

[35] P. 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*. Springer, 2011.

[36] A. C.-C. 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.



# A Quadratic-Residue based PKCR Encryption

To keep this paper self-contained, in this section we will go over the results from [27]. Specifically, we will be proving that the algorithms we referenced in section 3.1.3, re-randomization and proxy-reencryption, are correct and sound. This will finish the proofs in section 3.1.3, showing that Cocks' public-key encryption scheme is a PKCR encryption scheme.

## A.1 The QR Assumption, Jacobi Symbol, and Other Preliminaries

Before going into the re-randomization and re-encryption algorithms, we will first explain some notation and the assumption used in Cocks' scheme. First, let $\mathbf{QR}_N$ be the set of quadratic residues mod $N$, so

$$\mathbf{QR}_N := \{a \in \mathbb{Z}_N^* : \exists b \text{ s.t. } b^2 \equiv a \mod N\}.$$

Next, we use the standard notation for the Jacobi symbol: $\left(\frac{a}{N}\right) \in \{\pm 1\}$, which is polynomial-time computable via the law of Quadratic Reciprocity, and has the property that is multiplicative. So, for $N = pq$ (primes $p$ and $q$),

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{a}{q}\right) \text{ and } \left(\frac{ab}{N}\right) = \left(\frac{a}{N}\right) \cdot \left(\frac{b}{N}\right).$$

Where $\left(\frac{a}{p}\right) = 1$ (called the Legendre symbol because $p$ is prime) if and only if $a \in \mathbf{QR}_p$ (same for $q$).

The QR assumption states that it is hard to tell if an element in $\mathbb{Z}_N$ with Jacobi symbol 1 is a square or not. That is, we cannot tell if $a \in \mathbb{Z}_N$ is both a square mod $p$ and mod $q$ or is neither a square mod $p$ or $q$.

**Definition A.1.** The *Quadratic Residuosity Assumption* states that for all PPT adversaries $\mathcal{A}$, if $\mathcal{A}$ is given a composite RSA modulus $N = pq$, and an element $a \in \mathbb{Z}_N^*$ such that $\left(\frac{a}{N}\right) = 1$, then

$$\left|\Pr\left[\mathcal{A}(N, a) = 1 : a \in \mathbf{QR}_N\right] - \Pr\left[\mathcal{A}(N, a) = 1 : a \notin \mathbf{QR}_N\right]\right| < \text{negl}(|N|).$$

### A.1.1 More Notation and the Ring of Ciphertexts

Linear function ciphertexts will be critical in proving that we can re-randomize Cocks' ciphertexts as well as get AddLayer and DelLayer functionality.

Within $\mathbb{Z}_N$, we let $\mathbb{J}_1$ denote elements with Jacobi symbol 1 and $\mathbb{J}_{-1}$ denote the elements with Jacobi symbol $-1$. Squares mod $N$ are $\mathbf{QR}_N$ and squares mod $p$ are $\mathbf{QR}_p$.

**Linear Function Ciphertexts** Recall from section 3.1.3 given a modulus $N$ and public key $R \in \mathbf{QR}_N$, an encryption of a message $m \in \{0, 1\}$ is choosing a random $t \in \mathbb{Z}_N$ such that $\left(\frac{t}{N}\right) = (-1)^m$ and setting the ciphertext $c = t + Rt^{-1}$. Decryption is computing $\left(\frac{2r+c}{N}\right) = (-1)^{m'}$. Recall that the reason this worked was because $2r + t + Rt^{-1} \equiv t^{-1}(t + r)^2 \pmod{x^2 - R}$.

So, instead of having our ciphertext be represented by $2x + c$ (where we only need $c$ to be the ciphertext since $2x$ is assumed), we can actually have a ciphertext be any linear function $ax + b$ such that the decryption $\left(\frac{ar+b}{N}\right) = (-1)^m$, which was done in the work of Clear, Hughes, and Tewari [10]. The downside is that the ciphertexts have doubled in length, but the plus side is that we get an easy way to compute homomorphisms. For two ciphertexts $a_1x + b_1$ and $a_2x + b_2$ encrypting $m_1$ and $m_2$ respectively, $a_3x + b_3 = (a_1x + b_1) \cdot (a_2x + b_2) \pmod{x^2 - R}$ is the homomorphic addition of the two ciphertexts because when we decrypt, we have

$$\left(\frac{a_3r + b_3}{N}\right) = \left(\frac{(a_1r + b_1)(a_2r + b_2) + h(r)(r^2 - R)}{N}\right) = \left(\frac{a_1r + b_1}{N}\right) \cdot \left(\frac{a_2r + b_2}{N}\right) = (-1)^{m_1 \oplus m_2}.$$

What we've just done is multiply two elements in the ring $\mathbb{Z}_N[x]/(x^2 - R)$, and we have shown that our decryption algorithm respects this multiplication as homomorphic addition.

**Rings of Ciphertexts** As just shown, we will be working heavily in the ring $\mathbb{Z}_N[x]/(x^2 - R)$. We will introduce some notation for this ring and its counterparts. Let $\mathcal{R}_N = \mathbb{Z}_N[x]/(x^2 - R)$, $\mathcal{R}_p = \mathbb{Z}_p[x]/(x^2 - R)$, and $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^2 - R)$. To denote the multiplicative groups of these rings, we write $\mathcal{R}_N^*$, $\mathcal{R}_p^*$, and $\mathcal{R}_q^*$. Note that $|\mathcal{R}_N^*| = |\mathcal{R}_p^*| \cdot |\mathcal{R}_q^*| = (p^2 - 2p + 1)(q^2 - 2q + 1)$. So $\mathcal{R}_N^*$ is the overwhelmingly large fraction or $\mathcal{R}_N$, which has order $p^2q^2$; almost all elements in $\mathcal{R}_N$ have multiplicative inverses, and we can show that if $ax + b$ does not have an inverse, then either $a/b = r$ or $a^2R + b$ has a common factor with $N$ (so, either we violate the QR assumption, or we factor $N$).

A ciphertext $ax + b$ decrypts to 1 if $\left(\frac{ar+b}{N}\right) = 1$ and to $-1$ if $\left(\frac{ar+b}{N}\right) = -1$. We denote the set of ciphertexts in $\mathcal{R}_N^*$ that decrypt to 1 as $C_1$ and those that decrypt to $-1$ to be $C_{-1}$. Note that when using $C_1$ and $C_{-1}$, we are only referring to ciphertexts in the multiplicative group $\mathcal{R}_N^*$ (since getting anything without an inverse would imply breaking the QR assumption or factoring $N$).

#### Working Modulo a Prime $p$

In proving that we properly re-randomize a ciphertext, we will be relying on the *Chinese Remainder Theorem (CRT)*, and so understanding how $\mathcal{R}_p$ (and also $\mathcal{R}_q$) behave is important.

**Lemma A.2.** *If $ar + b$ is a square mod a prime $p$, then we can write $ax + b$ as $(cx + d)^2$ in $\mathcal{R}_p^*$. If $ar + b$ is a non-square mod a prime $p$, then we can take any $u \notin \mathbf{QR}_p$ and write $ax + b$ as $u(cx + d)^2 \pmod{x^2 - R}$.*

*Proof.* Let $u \in \mathbb{Z}_p^*$, but $u \notin \mathbf{QR}_p$.

All squares in $\mathcal{R}_p^*$ can be written as $(cx+d)^2$. Notice that $u(cx+d)^2$ cannot be a square in $\mathcal{R}_p^*$. Since squares account for exactly half of $\mathcal{R}_p^*$, we can write all non-squares as $u(cx+d)^2 - u$ as a member of the group is a bijection as a multiplicative map from $\mathcal{R}_p$ to itself, and thus $u(cx+d)^2$ maps to a different non-square for each square $(cx+d)^2$.

For a contradiction, assume $ar+b \in \mathbf{QR}_p$, but $ax+b$ is not a square. This means $ax+b = u(cx+d)^2 \pmod{x^2 - R}$ for some $cx+d$. But, when we evaluate at $r$, $(ar+b) = u(cr+b)^2$,

$$\left(\frac{u(cr+b)^2}{p}\right) = \left(\frac{u}{p}\right) \cdot \left(\frac{cr+b}{p}\right)^2 = -1 \neq \left(\frac{ar+b}{p}\right).$$

This is a contradiction, and so $ar+b$ being a quadratic residue mod $p$ implies $ax+b$ is a sqaure in $\mathcal{R}_p^*$.

If we let $ar+b \notin \mathbf{QR}_p$. We get that $ax+b = u(cx+d)^2$ for the same reason: if $ax+b = (cx+d)^2$, then evaluation at $r$ results in a contradiction. □

**Working In $\mathcal{R}_N$**

**Lemma A.3.** *All $ax+b \in C_1$ are of the form $(a'x+b')^2/t$ where $\left(\frac{t}{N}\right) = 1$. Similarly, all linear function encryptions $ax+b \in C_{-1}$ are of the form $(a'x+b')^2/t$ where $\left(\frac{t}{N}\right) = -1$.*

*Proof.* First, assume $ax+b \in C_1$. We have two cases of decryption to deal with: one where $ar+b$ is a square in both $\mathbb{Z}_p$ and $\mathbb{Z}_q$ and the other where it is a square in neither.

- $ar+b \in \mathbf{QR}_p$ and $\mathbf{QR}_q$. From lemma A.2, we know that $ax+b$ is a square in both $\mathcal{R}_p^*$ and $\mathcal{R}_q^*$. This means that mod $p$, we can write $ax+b = (c_p x + d_p)^2$ and mod $q$, $ax+b = (c_q x + d_q)^2$. By the Chinese Remainder Theorem (CRT) , we can thus write $ax+b = (cx+d)^2$. Now let $\gamma \in \mathbb{Z}_N^*$,

$$ax+b = \frac{(\gamma cx + \gamma d)^2}{\gamma^2} = \frac{(a'x+b')^2}{\gamma^2},$$

  and $\gamma^2$ is guaranteed to have Jacobi symbol 1 because it is a square.

- $ar+b \notin \mathbf{QR}_p$ or $\mathbf{QR}_q$. Again from lemma A.2, $ax+b$ is neither a square in $\mathcal{R}_p^*$ or $\mathcal{R}_q^*$. Again by CRT and lemma A.2, $ax+b = u(cx+d)^2$ where $u \in \mathbb{J}_1$ and $u \notin \mathbf{QR}_N$. Now, we can do the same trick as before, to get

$$ax+b = \frac{(\gamma cx + \gamma d)^2}{\gamma^2/u} = \frac{(a'x+b')^2}{\gamma^2/u}.$$

  Notice that $\gamma^2/u$ is neither a square mod $p$ or mod $q$ and thus has Jacobi symbol 1 mod $N$.

The case where $ax+b \in C_{-1}$ is similar. Now, without loss of generality, we can assume $ar+b \in \mathbf{QR}_p$ and $ar+b \notin \mathbf{QR}_q$. Lemma 3 tells us that mod $p$, $ax+b = 1 \cdot (c_p x + d_p)^2 \pmod{p}$ and $ax+b = u_q(c_q x + d_q)^2 \pmod{q}$. By the Chinese Remainder Theorem, we can find a $t \in \mathbb{Z}_N$ so that $t \equiv 1 \pmod{p}$ and $t \equiv 1/u_q \pmod{q}$, as well as $a' \equiv c_p \pmod{p}$ and $c_q \pmod{q}$ and $b' \equiv d_p \pmod{p}$ and $d_q \pmod{q}$. We can rewrite

$$ax+b \equiv_N (a'x+b')^2/t \pmod{x^2 - R}.$$

Now we have $\left(\frac{t}{N}\right) = \left(\frac{1}{p}\right) \cdot \left(\frac{u_q}{q}\right) = -1$ as desired. □

## A.2 Re-randomization of Cocks' Ciphertexts

We now have a method for getting the correct decryption of homomorphically added linear-function ciphertexts using the linear-function representation (as in Clear, Hughes, and Tewari [10]). Now we need to convert a linear-function ciphertext into a proper Cocks' ciphertext.

### A.2.1 Converting a Linear-function Ciphertext to a Cocks' Ciphertext

We define the following helper function, Convert, which takes in two elements $a$ and $b$ of $\mathbb{Z}_N^*$, a public key $R$, and returns a proper Cocks ciphertext that will decrypt to the same message as $\left(\frac{ar+b}{N}\right)$.

**function** QR.Convert$(a, b, R)$
  **while** $\left(\frac{2/a}{N}\right) \neq 1$ **do**
    $d, e \xleftarrow{\$} \mathbb{Z}_N$ and $t \xleftarrow{\$} \mathbb{J}_1$.
    $ax + b \leftarrow t^{-1}(dx + e)^2 \cdot (ax + b) \pmod{x^2 - R}$.
  **end while**
  **return** $c = 2b/a$
**end function**

**Claim A.4** ([27])**.** QR.Convert$(a, b, R)$ *outputs $c$ such that* $\left(\frac{ar+b}{N}\right) = \left(\frac{2r+c}{N}\right)$ *in polynomial time.*

*Proof.* There are two things to prove: first that Convert is correct and second that it terminates in polynomial time.

**Correctness of** QR.Convert Correctness is easy. Notice from our computation in the previous section, $ax+b$ decrypts to the same element after each loop. Now, once $\left(\frac{2/a}{N}\right) = 1$, then when we return $c = 2a/b$, we are really computing $(2/a)(ax+b) = 2x+2b/a$ and returning the constant term. So, when we decrypt $c$ like a Cocks ciphertext, we get

$$\left(\frac{2r + c}{N}\right) = \left(\frac{2r + 2a/b}{N}\right) = \left(\frac{2/a}{N}\right) \cdot \left(\frac{ar + b}{N}\right) = \left(\frac{ar + b}{N}\right),$$

which is exactly the decryption of the linear function ciphertext.

**Runtime of** QR.Convert**.** Time is a bit less straightforward, but as long as $ax + b$ has an inverse in the $\mathcal{R}_N$, we can prove that with with constant probability, we will find $d, e$ such that the resulting $a$ has $\left(\frac{2/a}{N}\right) = 1$. Recall that only a negligible fraction of elements $ax + b \in \mathcal{R}_N$ do *not* have inverses (and finding non-invertible elements breaks the cryptosystem), and so we can expect Convert never to deal with that case. We will show that if $ax + b \in \mathcal{R}_N$ has a multiplicative inverse, then

$$\Pr_{c,d \xleftarrow{\$} \mathbb{Z}_N^*, t \xleftarrow{\$} \mathbb{J}_1}\left[\left(\frac{a'}{N}\right) = 1 \text{ where } a'x + b' = (ax + b)(dx + e)^2 t^{-1}\right] = \frac{1}{2}.$$

And therefore $\Pr[\left(\frac{a/2}{N}\right) = 1] = \frac{1}{2}$ after each loop. So, we are only expected to loop two times.

We will be using the machinery and notation from section A.1.1, re-writing $ax + b$ in terms of being a square in $\mathcal{R}_p$ and $\mathcal{R}_q$ or not, and then going through the cases.

First, we know that we can rewrite $ax + b = (a'x + b')^2 t'^{-1}$ where $\left(\frac{t'}{N}\right) = \left(\frac{ar+b}{N}\right)$ by lemma A.3. Now, the term in our probability $(ax+b)\left((cx + d)^2 t^{-1}\right)$ becomes $((a'x+b')(dx+e))^2 (tt')^{-1}$. Since $a'x+b'$ will also have an inverse in $\mathcal{R}_N$, and we are choosing $dx + e$ at random, we are just as likely to choose $c, d$ as we are to choose $c', d'$ where $(c'x + d') = (a'x + b')(dx + e) \pmod{x^2 - R}$. The term we are trying to bound the probability on is $\frac{(dx+e)}{t}$ where $\left(\frac{t'}{N}\right) = \left(\frac{ar+b}{N}\right)$.

Let $m = \left(\frac{ar+b}{N}\right)$, the decryption of our ciphertext. We now have that the probability we are looking at is

$$\Pr_{c,d \xleftarrow{\$} \mathbb{Z}_N^*, t \xleftarrow{\$} \mathbb{J}_1}\left[\left(\frac{e}{N}\right) = 1 \text{ where } ex + f = (ax + b)\frac{(dx + e)^2}{t}\right]$$

$$= \Pr_{c,d \xleftarrow{\$} \mathbb{Z}_N^*, t \xleftarrow{\$} \mathbb{J}_m}\left[\left(\frac{e}{N}\right) = 1 \text{ where } ex + f = \frac{(dx + e)^2}{t}\right].$$

Now, we expand $\frac{1}{t} \cdot (dx+e)^2 \pmod{x^2-R}$, our linear term is just $\frac{2}{t}cd$. We can analyze

$$\begin{aligned}
&\Pr_{c,d \xleftarrow{\$} \mathbb{Z}_N^*, t \xleftarrow{\$} \mathbb{J}_m}\left[\left(\frac{2cd/t}{N}\right)=1\right] = \sum_{\gamma \in \mathbb{J}_1} \Pr_{c,d \xleftarrow{\$} \mathbb{Z}_N^*, t \xleftarrow{\$} \mathbb{J}_m}\left[\frac{2cd}{t}=\gamma\right] \\
&= \sum_{\gamma \in \mathbb{J}_1} \sum_{T \in \mathbb{J}_m} \Pr_{t \in \mathbb{J}_m}[t=T] \sum_{C \in \mathbb{Z}_N^*} \Pr_{c \in \mathbb{Z}_N^*}[c=C] \cdot \Pr_{d \in \mathbb{Z}_N^*}\left[d=\frac{T\gamma}{2C}\right] \\
&= \sum_{\gamma \in \mathbb{J}_1} \sum_{T \in \mathbb{J}_m} \frac{2}{\phi(N)} \sum_{A \in \mathbb{Z}_N^*} \frac{1}{\phi(N)} \cdot \frac{1}{\phi(N)} \\
&= \frac{\phi(N)}{2} \cdot \left(\frac{\phi(N)}{2}\frac{2}{\phi(N)}\right)\left(\phi(N) \cdot \frac{1}{\phi(N)}\right) \cdot \frac{1}{\phi(N)} = \frac{1}{2}.
\end{aligned}$$

□

### A.2.2 Using Convert to re-randomize

Now that we have QR.Convert, we can define the three algorithms for PKCR: randomization, adding a layer, and removing a layer. First, given any ciphertext and public key, the re-randomizing algorithm will produce a ciphertext computationally indistinguishable from a randomly generated fresh ciphertext of the same message.

**function** QR.Rand$(c, R)$
  $d, e \xleftarrow{\$} \mathbb{Z}_N$
  Let $t \xleftarrow{\$} \mathbb{Z}_N$ so that $\left(\frac{t}{N}\right) = 1$
  Compute $a'x + b' = (ax+b)(dx+e)^2 t^{-1} \pmod{x^2-R}$
  **return** QR.Convert$(a, b, R)$
**end function**

**Lemma A.5** ([27])**.** QR.*Rand*$(c, R)$ *for an encryption $c$ of a message $m$ outputs a $c'$ statistically (and therefore computationally) indistinguishable from a fresh encryption of $m$.*

*Proof.* We will show that choosing $d, e$, and $t$ statistically randomizes $2x+c$ as a ciphertext $ax+b$ in $C_m$. Once we have this, QR.Convert turns $ax+b$ into a random ciphertext in $C_m$ where $\left(\frac{a/2}{N}\right) = 1$, meaning the resulting Cocks ciphertext will be random in the space of $C_m$ of the form $2x + c'$.

So, let's show that choosing $d, e, t$ results in statistically randomizing $2x + c$in $C_m$. First, let's define the distribution $\mathcal{D}_{ax+b}$ for an element $ax+b$ in $\mathcal{R}_N$,

$$\mathcal{D}_{ax+b} := \left\{(ax+b)\frac{(dx+e)^2}{t} : dx+e \xleftarrow{\$} \mathcal{R}_N^*, t \xleftarrow{\$} \mathbb{J}_1\right\},$$

which is statistically close to the output from the third line in QR.Rand (when we compute $a'x+b'$). We say statistically close because $d$ and $e$ are chosen randomly from $\mathbb{Z}_N$. When it is obvious what $ax+b$ is (it will be $2x+c$, the linear function version of the Cocks' ciphertext we start with unless otherwise specified), we will drop the subscript, writing $\mathcal{D}$.

With overwhelming probability, $dx+e$ will have a multiplicative inverse in $\mathcal{R}_N^*$. So, we will prove that $\mathcal{D}$ randomizes ciphertexts that have inverses, and because the distribution from $\mathcal{D}$ is statistically close to the actual output of Rand and the set of ciphertexts that have inverses is statistically close to the set of all ciphertexts, Rand statistically re-randomizes a ciphertext.

Recall that our goal is to show that if $2x + c \in C_1$, then

$$\mathcal{D} \equiv \left\{fx+g \text{ where } fx+g \xleftarrow{\$} C_1\right\},$$

and if $2x + c \in C_{-1}$, then

$$\mathcal{D} \equiv \left\{fx+g \text{ where } fx+g \xleftarrow{\$} C_{-1}\right\}.$$

We will define an alternative distribution. Let $m = \left(\frac{ar+b}{N}\right)$, and let $\mathcal{D}'$ be

$$\mathcal{D}' := \left\{(dx+e)^2/t : dx+e \xleftarrow{\$} C_1, t \xleftarrow{\$} \mathbb{J}_m\right\}.$$

We will first show that $\mathcal{D} \equiv \mathcal{D}'$. We can rewrite $2x+c = (a'x+b')^2 t'^{-1}$ for some $t$ with Jacobi symbol 1 using lemma A.3. So, expanding our output from $\mathcal{D}$, $(2x+c) = ((\hat{a}x+\hat{b})(dx+e))^2(tt')^{-1}$. Since we are choosing $(dx+e)$ from the multiplicative group $\mathcal{R}_N^*$ at random and $\hat{a}x+\hat{b}$ is also in $\mathcal{R}_N^*$, the probability of $\mathcal{D}$ chooses $(dx+e)$ for its output is the same as the probability $\mathcal{D}$ chooses $(\hat{a}x+\hat{b})(dx+e)$ (mod $x^2-R$). Since $\left(\frac{tt'}{N}\right) = \left(\frac{t}{N}\right)$, the probability $\mathcal{D}$ outputs a specific $\frac{(dx+e)^2}{t}$ with $\left(\frac{t}{N}\right) = m$ is equivalent to the probability $\mathcal{D}'$ outputs that element in $\mathcal{R}_N^*$.

Our goal now is to show that $\mathcal{D}'$ outputs, uniformly, a ciphertext that decrypts to $m$. Note that, by counting, $|C_1| = |\mathcal{R}_N^*|/2$, since exactly half of the elements in $\mathcal{R}_N^*$ are squares divided by elements of Jacobi symbol 1 and the other half are squares divided by elements of Jacobi symbol $-1$. This means the probability that a uniform distribution on $C_1$ outputs $fx+g$ is $\frac{2}{|\mathcal{R}_N^*|}$.

Let $fx+g \in C_1$. We will analyze the probability $\mathcal{D}'$ outputs $fx+g$. By lemma A.3, $fx+g = \frac{(e'x+f')^2}{\gamma}$ where $\gamma \in \mathbb{J}_1$. We have two cases, $\gamma \in \mathbf{QR}_N$ and $\gamma \notin \mathbf{QR}_N$:

- $\gamma \in \mathbf{QR}_N$. The probability that $\mathcal{D}'$ outputs $fx+g$ is the probability that $\frac{(dx+e)^2}{t} = \frac{(f'x+g')^2}{\gamma}$ when we randomly choose $dx+e \in \mathcal{R}_N^*$ and $t \in \mathbb{J}_1$:

$$\begin{aligned}&\Pr_{dx+e \xleftarrow{\$} \mathcal{R}_N^*, t \xleftarrow{\$} \mathbb{J}_1}\left[\frac{(dx+e)^2}{t} = \frac{(f'x+g')^2}{\gamma}\right]\\ &= \sum_{T \in \mathbb{J}_1} \Pr_{t \xleftarrow{\$} \mathbb{J}_1}[t=T] \cdot \Pr_{dx+e \xleftarrow{\$} \mathcal{R}_N^*}\left[(dx+e)^2 = \frac{T}{\gamma}(f'x+g')^2\right].\end{aligned}$$

  Notice that this equation only has a solution in $dx+e$ if $t \in \mathbf{QR}_N$. Otherwise, we are trying to solve $(dx+e)^2 = \frac{t}{\gamma}(e'x+f')^2 \pmod{x^2-R}$ when $(dx+e)^2$ is a square, but $\frac{t}{\gamma}(f'x+g')^2$ is not.

  Now, assuming that $T/\gamma \in \mathbf{QR}_N$, we can let $k^2 = T/\gamma$ and rewrite $k^2(f'x+g'^2) = (kf'x+kg')^2 = (\hat{f}x+\hat{g})^2$. We are looking for the probability that a random $dx+e$ is a solution to $(dx+e)^2 = (\hat{f}x+\hat{g})^2$ (mod $x^2-R$). We need to know how many solutions there are to this. We will use CRT.

  Mod $p$, there are exactly two solutions $dx+e \in \mathcal{R}_p^*$ to $(\hat{f}x+\hat{g})^2$: at least two because $\pm(\hat{f}x+\hat{g})$ are both solutions, and no more than two because the size of squares is exactly half of $\mathcal{R}_p^*$. Mod $q$ there are also exactly two solutions. This means, mod $N$ there are 4 total solutions. Now, when we continue to analyze this probability, we have

$$\begin{aligned}&\sum_{T \in \mathbb{J}_1} \Pr_{t \xleftarrow{\$} \mathbb{J}_1}[t=T] \cdot \Pr_{dx+e \xleftarrow{\$} \mathcal{R}_N^*}\left[(dx+e)^2 = (T/\gamma)(f'x+g')^2\right]\\ &= \sum_{T \in \mathbf{QR}_N} \frac{2}{\phi(N)} \cdot \frac{4}{|\mathcal{R}_N^*|}\\ &= \left(\frac{\phi(N)}{4} \cdot \frac{2}{\phi(N)}\right) \cdot \frac{4}{|\mathcal{R}_N^*|}\\ &= \frac{2}{|\mathcal{R}_N^*|}.\end{aligned}$$

  So, in this case, the distribution $\mathcal{D}'$ is the same as uniform.

- $\gamma \notin \mathbf{QR}_N$. We can use the same analysis tricks, except $T$ must also not be in $\mathbf{QR}_N$, but still must

have Jacobi symbol 1. So,

$$\begin{aligned}
&\Pr_{dx+e\xleftarrow{\$}\mathcal{R}_N^*, t\xleftarrow{\$}\mathbb{J}_{-1}}\left[(dx+e)^2/t = (f'x+g')^2/\gamma\right] \\
&= \sum_{T\notin \mathbf{QR}_N} \frac{2}{\phi(N)}\cdot\frac{4}{|\mathcal{R}_N^*|} \\
&= \left(\frac{\phi(N)}{4}\cdot\frac{2}{\phi(N)}\right)\cdot\frac{4}{|\mathcal{R}_N^*|} \\
&= \frac{2}{|\mathcal{R}_N^*|}.
\end{aligned}$$

The case where $2x + c, fx + g \in C_{-1}$ is proved in exactly the same manner. □

### A.3 Adding and Deleting layers from a Cocks' ciphertext

The strategy here is taken from LaVigne's work in proxy-reencryption [27]. At a high level, $c$ is a valid ciphertext for a message $m$ encrypted under a public key $\mathsf{pk}$ if $\left(\frac{c\mathsf{sk}+2}{N}\right) = (-1)^m$. So, if we want to decrypt $c$ (encryption of $m$ under $\mathsf{pk}$) with a combined secret key $\mathsf{sk}\cdot\mathsf{sk}'$ (corresponding to public key $(\mathsf{sk}\cdot\mathsf{sk}')^2$) instead of $\mathsf{sk}$, we let $c' = c/\mathsf{sk}'$, and then decryption is guaranteed to produce the same output. Of course $c'$ is not a random-looking ciphertext, so then we apply $\mathsf{QR.Rand}$ to get our output.

**function** $\mathsf{QR.AddLayer}(c, \mathbf{k}', \mathsf{sk})$
  $c' \leftarrow c(\mathsf{sk})^{-1}$, and $R' \leftarrow \mathbf{k}'\cdot(\mathsf{sk}_2)^2$.
  **return** $\mathsf{QR.Rand}(c', R')$
**end function**
**function** $\mathsf{QR.DelLayer}(c, \mathbf{k}', \mathsf{sk})$
  $c' \leftarrow c(\mathsf{sk})$ and $R' = \mathbf{k}'/(\mathsf{sk}_2)^2$.
  **return** $\mathsf{QR.Rand}(c', R')$
**end function**

**Lemma A.6.** *For any public key* $\mathbf{k}'$ *and* $(\mathsf{pk}, \mathsf{sk})$ *in Cocks, and* $c = \mathsf{QR.Enc}(\mathbf{k}, m)$*, the following are equivalent distributions:*

$$\{\mathsf{QR.}\mathit{AddLayer}(c, \mathsf{sk})\} \equiv_s \{\mathsf{QR.Enc}(\mathbf{k}'\cdot\mathsf{pk}, m)\}$$

*and for* $c' = \mathsf{Enc}(\mathbf{k}'\cdot\mathsf{pk}, m)$*,*

$$\{\mathsf{QR.}\mathit{DelLayer}(c', \mathsf{sk})\} \equiv_s \{\mathsf{QR.Enc}(\mathbf{k}', m)\}$$

*Proof.* First, we will analyze $\mathsf{QR.AddLayer}$. The combined public key for adding a layer is $\mathbf{k}'\cdot\mathsf{pk} = (\mathsf{sk}'\cdot\mathsf{sk})^2$, with corresponding combined secret key $\mathsf{sk}'\cdot\mathsf{sk}$; and we define a ciphertext $c$ as a valid encryption of $m$ under public key $\mathsf{pk}$ if decryption results in $(-1)^m$, i.e. $\left(\frac{c\mathsf{sk}+2}{N}\right) = (-1)^m$.

The input to $\mathsf{QR.Rand}$ is the ciphertext $c'$ under public key $R' = \mathbf{k}'\cdot(\mathsf{sk})^2 \in \mathbf{QR}_N$. Let the secret key for $\mathbf{k}'$ be $\mathsf{sk}'$ (the square-root of $\mathbf{k}'$). Notice that decryption of the ciphertext $c$ is just taking the symbol of $c\mathsf{sk}' + 2$. Now, if we plug in the combined secret key $\mathsf{sk}'\cdot\mathsf{sk}$, we get that $c'\mathsf{sk}'\cdot\mathsf{sk} + 2 = c\mathsf{sk}' + 2$, and so the decryption is the same: $\left(\frac{c'\mathsf{sk}'\cdot\mathsf{sk}+2}{N}\right) = \left(\frac{c\mathsf{sk}'+2}{N}\right)$. Thus, $c'$ is a valid ciphertext encrypting the same message as $c$ under the combined public key $\mathbf{k}'\cdot(\mathsf{sk})^2 = \mathbf{k}'\cdot\mathsf{pk} = R'$. Then, when we apply $\mathsf{QR.Rand}$ to $c'$, the output is a ciphertext statistically indistinguishable from a fresh encryption of that message under the key $R'$.

We can go through the same logic for $\mathsf{QR.DelLayer}$. The un-combined public and secret keys are $\mathbf{k}$, and $\mathsf{sk}$ The input to $\mathsf{QR.Rand}$ is the ciphertext $c' = c\cdot\mathsf{sk}_2$, and so the decryption of $c'$ under the un-combined key is $\left(\frac{c'\mathsf{sk}'+2}{N}\right) = \left(\frac{c\mathsf{sk}'\cdot\mathsf{sk}+2}{N}\right)$. This makes $c'$ a valid encryption of the same message of $c$ under public key $\mathbf{k}'$. Since $\mathsf{QR.Rand}$ produces a ciphertext that is statistically indistinguishable from a fresh encryption of the same message, the output from $\mathsf{DelLayer}$ is a ciphertext indistinguishable from a fresh encryption of the message under the public key $\mathbf{k}'$. □

# B Compiling 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 section, 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 section 4.

## B.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.

### B.1.1 MPC model: Semi-honest adversaries

The material in this section is referenced one of the MPC models described by Goldreich in [14].

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 $\Pi$ denote some protocol for $n$ parties. $\Pi$ 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 $P_1, \ldots, P_n$, $\Pi$ *realizes* the functionality of $f$ if by the end of the protocol, each party $i$ gets the output $f(\mathbf{x})_i$.

**Definition B.1.** For a protocol $\Pi$ 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 B.2.** For a protocol $\Pi$ realizing a functionality $f$, we say $\Pi$ *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 corrupting some subset $I$ of the parties, but follows the protocol $\Pi$ (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.

### B.1.2 Topoloyg-Hiding Computation Model

Here we will review what it takes for a protocol to be topology hiding. The formal definition, from definition 4.1, 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 [7].

### B.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 B.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 B.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 $b = b'$.

## B.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 $\Pi$. 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 B.5** (Compiling THC from THB and PKE)**.** *Assume UC-secure THB and CPA-secure PKE exist. Then, for any PPT protocol* $\Pi$ *that privately computes a function* $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$*, there exists a PPT protocol* $\Pi'$ *that is a topology-hiding realization of the functionality of* $f$*.*

*Proof.* $\Pi$ 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. $\Pi'$ will operate as $\Pi$ except there will be a setup phase and point-to-point communication will be handled with broadcast and public-key encryption.

Let $\phi$ be the topology-hiding broadcast protocol. $\Pi'$ 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 $\phi$.

- Point-to-point communication. If party $i$ needs to send a message $m$ to party $j$ in protocol $\Pi$, $\Pi'$ dictates the following. First, party $i$ computes $c_i \leftarrow \mathsf{Enc}(pk_j, m)$. Then, party $i$ broadcasts $c_i$ using $\phi$ 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 $\Pi$.

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

Proving that this is topology-hiding is more involved. Since $\Pi$ 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 $\Pi'$ 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 $\Pi'$ is a topology-hiding MPC.

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

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

where each $R_i$ is actually a collection of messages representing the communication at round $i$ in the original protocol $\Pi$. 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 $\Pi'$.

- 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 $\mathsf{KeyGen}$ 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 $\mathsf{KeyGen}$. 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 $\phi$ 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 multiparty protocol $\Pi$, 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 $\Pi'$ 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 $\Pi$ 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 $\text{VIEW}_I$ 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 $\Pi$ privately computes $f$, so there exists a simulator $\mathcal{S}$ for $\Pi$. 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 $\Pi$.

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:

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

Therefore $\Pi'$ 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 B.6.** *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 $\Pi'$ that is a topology-hiding realization of the functionality of $f$.*

*Proof.* For every efficiently computable function $f$, there exists an MPC protocol $\Pi$ [36, 16, 15]. From theorem B.5, this means there exists a protocol $\Pi'$ which is the topology-hiding realization of $\Pi$. □