Topology-Hiding Computation on All Graphs

Adi Akavia[*] Rio LaVigne[†] Tal Moran[‡]

January 24, 2018

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-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 random-walks that, together, are locally identical to receiving at each round a message from each neighbors and sending back processed messages 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 [29, 13, 12]. 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 [17], the answer is mostly negative. The situation is better in the computational setting. Moran, Orlov and Richelson showed that topology-hiding computation *is* possible against static, semi-honest adversaries [25]; followed by constructions with improved efficiency that make only black-box use of underlying primitives [18]. However, all these protocol are restricted to communication graphs with *small diameter*.

[*]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.

Specifically, these protocols address networks with diameter $D = O(\log n)$, logarithmic in the number of nodes n (where the diameter is the maximal distance between two nodes in the graph). Akavia and Moran [1] showed that topology hiding computation is feasible also for *large diameter* networks of certain forms, most notably, cycles, trees, and low circumference graphs.

However, there are natural network topologies not addressed by the above protocols [25, 18, 1]. They include, for example, wireless and ad-hoc sensor networks (e.g. mesh networks for cellphones, etc), as in [9, 27]. The topology in these graphs is modeled by random geometric graphs [26], where, with high probability, the diameter and the circumference are simultaneously large [10, 2]. These qualities exclude the use of all aforementioned protocols. So, the question remained:

Is topology hiding MPC feasible for every network topology?

2 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 and semi-honest as in the prior works [25, 18, 1].[1] Our protocol also fits a stronger definition of security than that from prior works: instead of allowing the adversary to know who his neighbors are, he only gets pseudonyms; importantly, an adversary cannot tell if two nodes he controls share an honest neighbor.

Theorem 2.1 (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 5.3.

As in [25, 18, 1], given a *topology-hiding broadcast* for a point-to-point channels network, we can execute on top of it any MPC protocol from the literature that is designed for networks with broadcast channels; the resulting protocol remains topology-hiding. 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) [29, 13, 12], we conclude that *topology-hiding MPC* for all efficiently computable functionalities and all network topologies exists (assuming public key encryption exists).

Corollary 2.2. *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 corollary 5.4.

2.1 High-Level Overview of our Techniques

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

A challenge we face is that the walk itself may reveal topology information. For example, a party can deduce the graph commute-time from the number of rounds before a returning visit by the walk. We therefore hide the random walk by using multiple simultaneous random walks (details below). The combination of all our random walks obeys a simple communication structure: at every round each node

[1]Moran et al.[25] consider also a fail-stop adversary for proving an impossibility result.

receives an incoming message from each of its neighbors, randomly permutes the messages, and sends them back, one along each outgoing edge.

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

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

From the point of view of a node v with d neighbors, the random walks on the forward-phase are specified by choosing a sequence of independent random permutations $\pi_t \colon [d] \to [d]$, 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.

2.2 Related Work

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 [25]. 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 [1], which takes a different approach. They employ ideas from cryptographic voting literature, hiding the order of nodes in the cycle by "mixing" encrypted inputs before decrypting

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

Graphs families	[18, 25]	[1]	[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.

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 [18], which describes a protocol that achieves better efficiency than [25] (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 [6] 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.

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

Secure Multiparty Computation with General Interaction Patterns. Halevi, Ishai, Jain, Kushilevitz, and Rabin [15] presented a unified framework for studying secure MPC with arbitrarily restricted interaction patterns, generalizing models for MPC with specific restricted interaction patterns [14, 3, 16]. Their goal is not topology hiding, however. Instead, they ask the question of when is it possible to prevent an adversary from learning the output to a function on several inputs. They started by observing that an adversary controlling the final players $P_i, \cdots, P_n$ in the interaction pattern can learn the output of the computed function on several inputs because the adversary can rewind and execute the protocol on any possible party values $x_i, \dots, x_n$. This model allows complete knowledge of the underlying interaction pattern (or as in our case, the communication graph).

2.3 Organization of Paper

In sections 3.1 to 3.4 we describe our adversarial model and introduce definitions and our notation. In section 3.4 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 3.7, 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 3.7.1 we discuss how correlated random walks fit that description, and in section 3.7.2 we prove that universal exploration sequences also satisfy the description. 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.[18, 1]. In section 5, we explain our protocol for topology-hiding broadcast on general graphs and prove its completeness and security. Then, in section 5.3, we go over a time and communication tradeoff, and explain how we can optimize our protocol with respect to certain classes of graphs. Finally, in section 6, we conclude and discuss future work.

3 Preliminaries

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

We consider a static and computationally bounded (PPT) adversary that controls some subset of parties (any number of parties). That is, at the beginning of the protocol, the adversary corrupts a subset of the parties and may instruct them to deviate from the protocol according to the corruption model. Throughout this work, we consider only semi-honest adversaries. For general MPC definitions including in-depth descriptions of the adversarial models we consider, see [11].

3.2 Notation

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

3.2.1 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 $d_v = |\mathcal{N}(v)|$.

3.2.2 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 $pk^{(t)}_{i \to d}$).

3.3 UC Security

As in [25], we prove security in the UC model [4]. If a protocol is secure in the UC model, it can be composed with other protocols without compromising security, so we can use it as a subprotocol in other constructions. This is critical for constructing topology-hiding MPC based on broadcast—broadcast is used as a sub-protocol.

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

3.4 Privately Key-Commutative and Rerandomizable Encryption

As in [1], we require a public key encryption scheme with the properties of being *homomorphic* (with respect to OR in our case), *privately key-commutative*, and *re-randomizable*. In this section we first formally define the properties we require, and then show how they can be achieved based on the Decisional Diffie-Hellman assumption.

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

3.4.1 Required Properties

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

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

Privately Key-Commutative

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

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

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

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

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

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

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

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

Rerandomizable

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

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

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

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

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

Homomorphism

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

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

Homomorphic OR

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

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

Claim 3.1. *Let $\mathcal{M}$ have prime order p, where $1/p$ is negligible in the security parameter, and pk, sk be corresponding public and secret keys. For every two ciphertexts c, c such that $\mathsf{Dec}(c, sk) = M$ and $\mathsf{Dec}(c', sk) = M'$, the distribution $(c, c', pk, sk, M, M', \mathsf{Enc}(M \vee M', pk; U^*))$ is statistically indistinguishable from*
$(c, c', pk, sk, M, M', \mathsf{HomOR}(c, c', pk; U^*))$.

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

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

- Next, let c_0 be an encryption of 0 and c'_1 be an encryption of 1. In this case, 0 is represented again by $1_{\mathcal{M}}$, but c'_1 is represented by any $m' \leftarrow \mathcal{M}\,\{1_{\mathcal{M}}\}$ (otherwise c'_1 would decrypt to 0). After hPower, $\hat{c}_0$ still encrypts $1_{\mathcal{M}}$, but $\hat{c}'_1$ encrypts $\hat{m} = m'^{r'}$ for some $r' \overset{\$}{\leftarrow} \mathbb{Z}_p$. hMult yeilds an encryption of $\hat{m}$ and Rand makes a ciphertext computationally indistinguishable from a fresh encryption of $\hat{m}$. Since $\mathcal{M}$ has prime order p and $r' \overset{\$}{\leftarrow} \mathbb{Z}_p$, as long as $m' \neq 1$, m'^{r} is uniformly distributed over $\mathcal{M}$. Note that uniform over $\mathcal{M}$ and uniform over $\mathcal{M}\,\{1_{\mathcal{M}}\}$ are statistically indistinguishable distributions because there is a $1/p = \mathrm{negl}(\kappa)$ probability that uniform over $\mathcal{M}$ produces $1_{\mathcal{M}}$ — all other elements are produced with roughly the same uniform probability. Therefore, the distribution of the resulting homomorphically OR'd ciphertext is statistically indistinguishable from a fresh encryption of the boolean message 1 (which is an encryption of a uniform element over $\mathcal{M}\,\{1_{\mathcal{M}}\}$).

- Finally, let c_1 and c'_1 both be encryptions of 1: c_1 encrypts $m \overset{\$}{\leftarrow} \mathcal{M}$ and c'_1 encrypts $m' \overset{\$}{\leftarrow} \mathcal{M}$. We will go through the same steps to have at the end, a ciphertext computationally indistinguishable[3] from a fresh encryption of $m^r \cdot m'^{r'}$ for $r, r' \overset{\$}{\leftarrow} \mathbb{Z}_p$. Again because the order of $\mathcal{M}$ is prime, $m^r \cdot m'^{r'}$ is uniformly distributed over $\mathbb{Z}_p$, and so the resulting ciphertext looks like a fresh encryption of 1.

$\square$

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

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

3.5 Instantiation of OR-homomorphic PKCR-enc under DDH

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

```
function RAND($c = (c_1, c_2), pk; r$)
    return $(c_1 \cdot g^r, pk^r \cdot c_2)$
end function
```

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

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

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

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

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

```
function HMULT($c = (c_1, c_2), c' = (c'_1, c'_2)$)
    return $c'' = (c_1 \cdot c'_1, c_2 \cdot c'_2)$
end function
```

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

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

3.6 Instantiation of OR-homomorphic PKCR-enc under QR

We will be using an instantiation of Cocks' Identity-Based Encryption scheme (IBE) [8]. The scheme was shown to be homomorphic by Joye [19]. Joye's work was extended to show that the Cocks' scheme can be used for two-way proxy re-encryption by LaVigne [22]. Using the same techniques from LaVigne's work, we can show that this IBE can be compiled into an OR-homomorphic PKCR encryption scheme.

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. Here is a quick review of the public-key scheme. For details on the QR assumption and the Jacobi symbol, see appendix A.1.

- QR.Gen(1^κ) choose an κ-secure RSA modulus $N = pq$, $r \overset{\$}{\leftarrow} \mathbb{Z}_N^*$, and $R \leftarrow r^2 \pmod N$. Output the public-secret key pair $pk = (N, R)$ and $sk = r$.

- QR.Enc$(pk, m \in \{0,1\})$. Sample $t \overset{\$}{\leftarrow} \mathbb{Z}_N$ 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(sk, c). Compute $\left(\frac{2r+c}{N}\right) = (-1)^{m'}$. Output m'.

The reason decryption works is because $c + 2r \equiv (t + r)^2 \cdot t^{-1}$ (see appendix A.1 for details). Now, because the Jacobi symbol is multiplicative, we have

$$\left(\frac{c+2r}{N}\right) = \left(\frac{t^{-1}(t+r)^2}{N}\right) = \left(\frac{t^{-1}}{N}\right) \cdot \left(\frac{(t+r)}{N}\right)^2 = \left(\frac{t}{N}\right)^{-1} \cdot 1 = \left(\frac{t}{N}\right) = (-1)^m.$$

3.6.1 Getting an XOR-Homomorphic PKCR-enc under QR

From Joye and LaVigne, we have that this scheme is XOR-homomorphic[19, 22]. We still need to show that it is PKCR. To show this, we will be using the proxy-reencryption algorithms from LaVigne [22]. For completeness, full proofs will be included in appendix A.

First, we need randomness. LaVigne explicitly defined a re-randomizing algorithm: $\mathsf{QR.Rand}$.

Lemma 3.2 ([22]). *There is an efficient algorithm,* $\mathsf{QR}.\mathit{Rand}$*, such that when given a Cocks public key $pk = (N, R = r^2)$, and ciphertext $c = [m]_R$, outputs another encryption c' of m computationally indistinguishable from a fresh encryption of m under the same public key:*

$$\{\mathsf{QR}.\mathit{Rand}(N, R, [m]_R), c, r\} \equiv_c \{\mathsf{QR.Enc}(N, R, m), c, r\}.$$

Now, for the privately key-homomorphic part of PKCR we will employ LaVigne's two-way proxy re-encryption.

Lemma 3.3 ([22]). *There is an efficient algorithm,* $\mathsf{ReEncrypt}$*, when given a Cocks public key $pk = (N, R = r^2)$, a new public key in the same modulus $(R' = r'^2)$, a ciphertext $[m]_R$, and re-encryption key $T = r/r'$, outputs a ciphertext $c' = [m]_{R'}$ of the same message encrypted under the new public key R' that is computationally indistinguishable from a fresh encryption of m under R':*

$$\{\mathsf{ReEncrypt}(N, R, R', c, T), r'\} \rightarrow c' \equiv_c \{\mathsf{QR.Enc}(N, R', m), r'\}$$

Both of these lemmas, lemma 3.2 and lemma 3.3, are proved in section 5 of [22], and provided for completeness in the appendix.

Using lemma 3.3, we can get adding and removing layers essentially for free, using re-encryption keys $T = 1/r'$ and $T = r'$ respectively.

```
function AddLayer($c, pk = (N, R), sk' = r'$)
    Let $R' \leftarrow r'^2 \pmod N$.
    return ReEncrypt($N, R, RR', c, r'^{-1}$)
end function
function DelLayer($c = (c_1, c_2), pk = (N, R), sk' = r'$)
    Let $R' \leftarrow r'^2 \pmod N$.
    return ReEncrypt($N, R, RR'^{-1}, c, r'$)
end function
```

Theorem 3.4. *AddLayer and DelLayer are efficient and correct.*

Proof. First, $\mathsf{ReEncrypt}$ is efficient, correct, and secure from lemma 3.3. Now, let's start with adding a layer to a message $m \in \{0, 1\}$ encrypted under public key N, R: $[m]_R$. We have a secret key r' corresponding to the public key $R' = r'^2$. We want to end up with the ciphertext $[m]_{RR'}$ — where RR' is just the product mod N of the two public keys, and the ciphertext is indistinguishable from a fresh encryption of m under RR' (since R and R' are both squares, their product is also a square, and hence a valid public key); the corresponding secret key for RR' is rr', just the product of the two secret keys.

Using the language of re-encryption, we want to go from a message under public key R to being encrypted under RR'. The corresponding re-encryption key is $r/(rr') = r'^{-1} \pmod N$. Since we call $\mathsf{ReEncrypt}(N, R, RR', [m]_R, r'^{-1})$, the resulting ciphertext will be $[m]_{RR'}$ by correctness of $\mathsf{ReEncrypt}$, and it will be computationally indistinguishable from a fresh encryption of m under RR' from the soundness of $\mathsf{ReEncrypt}$.

We can use the same logic to prove that DelLayer is correct and secure. We want to go from a message under public key R to being encrypted under $RR'^{-1} \pmod{N}$. The re-encryption key in this case is $r/(rr'^{-1}) \equiv r' \pmod{N}$. Therefore, $\mathsf{ReEncrypt}(N, R, RR'^{-1}, [m]_R, r') = [m]_{RR'^{-1}}$ is correct and computationally indistinguishable from a fresh encryption of m under RR'^{-1} by lemma 3.3. □

Remark. Two-way proxy re-encryption and PKCR are fundamentally different. For example, two-way proxy re-encryption requires *both* secret keys to create the re-encryption key, and there are no guarantees about the relationships between public keys (e.g., there may not be a group structure). For PKCR, we require knowing one of the secret keys, so there is no notion of a re-encryption key that hides both keys. It is mostly a quirk of Cocks' IBE and the relationship between public keys in that setting that we can use the proxy re-encryption algorithm to add and remove layers of public keys.

3.6.2 Going from PKCR XOR-homomorphic to PKCR OR-homomorphic

Now, we have a scheme that is PKCR and XOR-homomorphic. We need to get an OR-homomorphism out of this—parity is linear and does not directly give us the non-linear functionality of OR. The general idea is the following: given an XOR-homomorphic scheme with security parameter κ, we can construct a new ciphertext consisting of κ of the original ciphertexts. We encrypt 0 by having κ encryptions of 0, and we encrypt 1 by encrypting a random bit in each of the κ ciphertexts. Because the scheme is XOR-homomorphic and randomizable, we can compute OR by homomorphically XOR-ing two of the ciphertext vectors component-wise. XOR'ing two ciphertext vectors of 0's, or if one of the ciphertext vectors is all 0's, then the resulting ciphertext vector will clearly be correct. Moreover, when they are both encryptions of 1, we can still argue that the resulting component-wise XOR will produce a vector of encoding 1 with probability at least $1 - 1/2^\kappa$.

Claim 3.5. *Any XOR-homomorphic PKCR encryption scheme can be compiled into an OR-homomorphic PKCR scheme.*

Proof. First, let Gen, Enc, Dec, XOR-Hom, Rand, AddLayer, DelLayer be the algorithms for the XOR-homomorphic PKCR encryption scheme. We will refer to the compiled OR-homomorphic scheme as an extension of the original XOR-homomorphic scheme.

For the sake of notation, for two encrypted values c, c', let $c'' = c + c'$ denote homomorphically XOR'ing the two ciphertexts, and by extension

$$\sum_{i=1}^{\ell} c_i := \mathsf{XOR\text{-}Hom}(c_1, \mathsf{XOR\text{-}Hom}(\ldots, \mathsf{XOR\text{-}Hom}(c_{\ell-1}, c_\ell, pk)\ldots), pk).$$

Likewise, $0 \cdot c$ or $1 \cdot c$ are valid linear homomorphic operations to apply to a ciphertext in our XOR-homomorphic setting:

$$0 \cdot c := \mathsf{Enc}(pk, 0) \qquad \text{and} \qquad 1 \cdot c := \mathsf{Rand}(pk, c).$$

So, using this notation, the corresponding algorithms for the extension are as follows:

- $\mathsf{Ext.Gen}(1^\kappa)$ outputs the public-secret key pair (pk, sk) generated by $\mathsf{Gen}(1^\kappa)$.
- $\mathsf{Ext.Enc}(pk, m \in \{0,1\})$

```
if $m = 0$ then
    For each $i \in [\kappa]$, $c_i \leftarrow \mathsf{Enc}(pk, 0)$.
else
    For each $i \in [\kappa]$, $b_i \xleftarrow{\$} \{0,1\}$.
    $c_i \leftarrow \mathsf{Enc}(pk, b_i)$.
end if
Output $\mathbf{c} = (c_1, \ldots, c_\kappa)$.
```

- $\mathsf{Ext.Dec}(sk, \mathbf{c})$

```
For each $i \in [\kappa]$, $m_i \leftarrow \mathsf{Dec}(sk, c_i)$.
**if** there exists $i$ such that $m_i = 1$ **then**
    Output 1.
**else**
    Output 0.
**end if**
```

- $\mathsf{Ext.Rand}(\mathbf{c}, pk)$.

```
$\mathbf{A} \xleftarrow{\$} \{0,1\}^{\kappa\times\kappa}$ such that $\mathbf{A}^{-1}$ exists in $\mathbb{Z}_2^{\kappa\times\kappa}$.
For each $i \in [\kappa]$, $c'_i \leftarrow \mathsf{Rand}(\sum_{j=1}^{\kappa} a_{i,j} \cdot c_j, pk)$.
Output $\mathbf{c}' = (c'_1, \ldots, c'_\kappa)$.
```

- $\mathsf{Ext.AddLayer}(\mathbf{c}, pk, sk')$

```
For each $i \in [\kappa]$, $c'_i \leftarrow \mathsf{AddLayer}(c_i, pk, sk')$.
Output $\mathbf{c}' = (c'_1, \ldots, c_\kappa)$.
```

- $\mathsf{Ext.DelLayer}(\mathbf{c}, sk', pk)$

```
For each $i \in [\kappa]$, $c'_i \leftarrow \mathsf{DelLayer}(c_i, pk, sk')$.
Output $\mathbf{c}' = (c'_1, \ldots, c_\kappa)$.
```

- $\mathsf{Ext.HomOR}(\mathbf{c}, \mathbf{c}', pk)$

```
$\hat{\mathbf{c}} \leftarrow \mathsf{Ext.Rand}(\mathbf{c}, pk)$
$\hat{\mathbf{c}}' \leftarrow \mathsf{Ext.Rand}(\mathbf{c}', pk)$
For each $i \in [\kappa]$, $c''_i \leftarrow \hat{c}_i + \hat{c}'_i$
Output $\mathbf{c}'' = (c''_1, \ldots, c''_\kappa)$.
```

Each of these algorithm clearly only takes time polynomial in κ as long as the origin algorithms also only take polynomial time in κ. So, we just need to prove correctness.

First, correctness of Ext.Enc and Ext.Dec is clear. When $m = 0$, decryption will always be correct—there will never be any 1's encrypted in the vector of ciphertexts. When $m = 1$, decryption is correct with probability $1 - \frac{1}{2^\kappa}$, so all but negligible chance that we decrypt correctly.

Now, correctness of Ext.Rand is a bit trickier. We are, in essence, computing the matrix multiplication $\mathbf{A} \cdot \mathbf{c}$ to get another encrypted vector $\mathbf{c}'$. Now, if $\mathbf{c}$ encrypts 0, then no matter what $\mathbf{A}$ is, the resulting vector of ciphertexts $\mathbf{c}'$ will be all 0's. If $\mathbf{c}$ encrypts 1 and decrypts correctly (so there is at least one 1), then because $\mathbf{A}$ is invertible mod 2, $\mathbf{c}'$ will also contain at least one 1, and decrypt correctly. So, Ext.Rand is correct with all but negligible probability.

Next, adding and removing layers is done component-wise. So, because the original AddLayer and DelLayer are correct, this extension will also be correct (adding and removing layers will not affect the values in the vectors themselves).

Finally, we will argue why Ext.HomOR is correct. We have three cases to deal with:

- $\mathbf{c}$ and $\mathbf{c}'$ both encrypt 0. In this case, both $\hat{\mathbf{c}}$ and $\hat{\mathbf{c}}'$ are both ciphertext vectors encrypting all 0's, so the homomorphic addition yields only encryptions of 0's. Therefore, $\mathbf{c}''$ is a correct ciphertext encrypting only 0's.

- One of $\mathbf{c}$ or $\mathbf{c}'$ encrypts 1 and the other encrypts 0. Note that the algorithm is symmetric with respect to $\mathbf{c}$ and $\mathbf{c}'$, so without loss of generality, assume $\mathbf{c}$ encrypts 1 and $\mathbf{c}'$ encrypts 0. Assume that $\mathbf{c}$ is a correct encryption of 1, so there is at least one non-zero coordinate. Randomizing $\mathbf{c}$ to get $\hat{\mathbf{c}}$ randomizes the location of non-zero coordinates. For $\mathbf{c}$, Ext.Rand yields a random 0-1 vector because we multiply by a random linear transformation $\mathbf{A}$ with the exception that it will not produce the all-0's vector because $\mathbf{A}$ is invertible. Therefore, XOR'ing with $\hat{\mathbf{c}}'$ (which is the all-0's vector) yields $\mathbf{c}''$, a random 0-1 vector that will correctly decrypt to 1 as long as $\mathbf{c}$ decrypts to 1 (which happens with probability $1 - \mathrm{negl}(\kappa)$).

- Both $\mathbf{c}$ and $\mathbf{c}'$ encrypt 1. Assume that at least one of $\mathbf{c}$ and $\mathbf{c}'$ decrypt correctly (happens with probability $(1 - \text{negl}(\kappa))$). We notice that Ext.Rand produces a random 0-1 vector for at least one of $\mathbf{c}$ or $\mathbf{c}'$, and therefore, XOR'ing their coordinates also produces a (uniformly) random 0-1 vector. Therefore, our output $\mathbf{c}''$ will correctly decrypt to 1 with probability $1 - \text{negl}(\kappa)$.

So, in all cases, Ext.HomOR correctly homomorphically OR's ciphertexts together with probability $1 - \text{negl}(\kappa)$. □

As shown in the last section, Cocks' public key scheme is an XOR-homomorphic PKCR-encryption scheme. This means we can extend it into an OR-homomorphic scheme with the previous construction. Thus, the QR assumption also yields an OR-homomorphic PKCR-encryption.

3.7 Graph Exploration Sequences

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

Sequence and walk will often be used interchangeably because a sequence induces a walk and vice-versa.

Definition 3.6. An *exploration sequence* for d-regular graphs on n nodes is a sequence $\tau_1, \dots, \tau_T \in \{0, \dots, d-1\}$ and starting edge $e_0 = (v_{-1}, v_0)$ so that the resulting walk $v_1, \dots, v_T$ covers all d-regular graphs on n nodes.

We can also define an exploration sequence for non-regular graphs, just based on the maximum degree d, or n for any graph. In this case, our walk is still just defined by a starting edge and offsets $\tau_1, \dots, \tau_T \in \{0, \dots, d\}$, but if we enter node v_i from edge s and node v_i has degree d_i, we take edge $s + \tau_i \mod d_i$ to node v_{i+1}.

We can consider an exploration sequence with errors as well. In this model, we consider the τ_i's are generated by some random process, and so there could be some probability that the walk fails.

Definition 3.7. An *δ-exploration sequence* for n-node graphs with maximum degree d is a sequence $\tau_1, \dots, \tau_T \in \{0, \dots, d-1\}$ and starting edge $e_0 = (v_{-1}, v_0)$ so that the resulting walk $v_1, \dots, v_T$ covers every n-node graph with max degree d with probability at least $1 - \delta$ over the randomness used to generate the sequence.[4]

To get correctness, we will need to run at least one of these walks per node. For topology-hiding, want to have one walk per direction on each edge (so a total of $2 \cdot |\text{edges}|$ walks). Moreover, we want this property for ever step in these simultaneous walks. So, for every direction of every edge, we want exactly one walk to be traveling down it at each step; we do not want the walks to "interfere" with each other. For instance, 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. The reason for this is that it makes it easier to describe and analyze our protocols—the topology of the graph does not interfere with the walks when define them to have this property.

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).

[4]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.

Definition 3.8. 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 3.9. 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 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; Seq is an *information-local function*.

Definition 3.10 ([1])**.** A function computed over a graph $G = (V, E)$ is *information-local* if the output of every node $v \in V$ can be computed from its own input and random coins.[5]

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 5.3.

Remark. If we have a full collection of non-interfering information-local exploration sequences, that can be computed given the exact number of nodes n in the graphs we are dealing with, then we also get a 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 stringing 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).

3.7.1 Correlated Random Walks

Here we will prove that correlated random walks of length $O(\kappa \cdot n^3)$ are an example of a collection of non-interfering $\text{negl}(\kappa)$-exploration sequences. By theorem 5.3, this will imply an instantiation of our protocol that uses random walks. In order to prove this, we will first need to prove some qualities about the random walks. We will rely on the following definition and theorem from Mitzenmacher and Upfal's book (see chapter 5)[24].

Definition 3.11 (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 3.12 (Cover time bound)**.** *The cover time of any connected, undirected graph $G = (u, v)$ is bounded above by $4nm \leq 4n^3$.*

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

Corollary 3.13. *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 3.12 tells us that we expect $t \leq 4n^3$, and so by a Markov bound, we have

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

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

We can represent $\mathcal{W}(u, \tau)$ as a union of τ random walks, each of length $8n^3$: $\mathcal{W}(u_1 = u, 1) \cup \mathcal{W}(u_2, 1) \cup \cdots \cup \mathcal{W}(u_\tau, 1)$, where u_i is the node we have reached at step $i \cdot 8n^3$ (technically, u_i is a random variable, but the specific node at which we start each walk will not matter). $\mathcal{W}(u, \tau)$ will succeed in covering all nodes in G if any $\mathcal{W}(u_i, 1)$ covers all nodes.

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

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

Therefore,

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

□

Lemma 3.14. *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 3.13, we also know that each walk has probability $2^{-\kappa} = \text{negl}(\kappa)$ of covering the entire graph. The lemma follows immediately. □

3.7.2 Perfect Covering: Universal Exploration Sequences

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

UESs are typically just described for d-regular graphs, but that is mostly because any general graph can be transformed into a 3-regular graph using a transformation by Koucky [21].

Definition 3.15. A *universal exploration sequence* for d-regular graphs of size n is a sequence of instructions $\tau_1, \ldots, \tau_T \in \{0, 1, \ldots, d-1\}$ so that if for every connected d-regular graph $G = (V, E)$ on n vertices, any number of its edges, and any starting edge $(v_{-1}, v_0) \in E$, then walk visits all vertices in the graph.

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

Lemma 3.16 ([28]). *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 3.17. *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 3.16, 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: τ_t. So, a walk entering from edge e will leave edge $e + \tau_t \mod d_i$. For two walks to collide, $e + \tau_t = e' + \tau_t \mod d_i$, implying $e = e' \mod d_i$. Since $0 \leq e, e' < d_i$, we get that $e = e'$, contradicting that no walks were interfering before this step. Therefore, none of these walks will interfere. □

4 A Stronger Simulation-based Definition of Topology-Hiding

Here we adapt the simulation-based definition of topology-hiding from [25] 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 [25]). Our definition, similar to [25], 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, [25] define the $\mathcal{F}_{\text{graph}}$-hybrid model, which employs a special "graph party," P_{graph}. Figure 4.1 shows $\mathcal{F}_{\text{graph}}$'s functionality: at the start of the functionality, $\mathcal{F}_{\text{graph}}$ receives the network graph from P_{graph}, and then outputs, to each party, that party's neighbors. Then, $\mathcal{F}_{\text{graph}}$ acts as an "ideal channel" for parties to communicate with their neighbors, restricting communications to those allowed by the graph.

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

In our definition, $\mathcal{F}_{\text{Graph}}$ recieves the graph from P_{graph} (as in [25]), but—unlike [25]—$\mathcal{F}_{\text{Graph}}$ 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 $(\mathcal{F}_{\text{graphInfo}} || \mathcal{F})$.

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

Definition 4.1. We say that a protocol Π is a topology-hiding realization of a functionality $\mathcal{F}$ if it UC-realizes $(\mathcal{F}_{\text{graphInfo}} || \mathcal{F})$ in the $\mathcal{F}_{\text{graph}}$-hybrid model.

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

Participants/Notation:
This functionality involves all the parties $P_1, \dots, P_n$ and a special graph party P_{graph}.

Initialization Phase:

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

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

Communication Phase:

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

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

Figure 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 $f(u, v) = f(v, u)$.

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

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

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

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

4.1 Differences of this Model: Neighbors of Neighbors

In the first model, proposed by [25], $\mathcal{F}_{\text{graphInfo}}$ 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.

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

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

5 Topology Hiding Broadcast protocol for General Graphs

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

The protocol (see protocol 1) is composed of two phases: an aggregate (forward) phase and a decrypt (backward) phase. In the aggregate phase messages traverse a walk (an exploration sequence, see definition 3.7) 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 $\pi_t^{-1}(j)$.

Protocol 1 Topology-hiding broadcast for general graphs. Inputs parameters: n is the number of nodes; $\text{negl}(\kappa)$ the failure probability; d_i the degree of node i; and b_i the input bit of node i. See section 3.2.2 for an explanation of notation.

```
1:  procedure BROADCAST($(n, \kappa, d_i, b_i)$)
2:      // The number of steps we take in our random walk will be $T$
3:      $T \leftarrow \kappa \cdot 8n^3$
4:      Generate $T \cdot d_i$ key pairs: for $t \in [T]$ and $d \in [d_i]$, generate pair $(pk^{(t)}_{i\to d}, sk^{(t)}_{i\to d}) \leftarrow \mathsf{KeyGen}(1^\kappa)$.
5:      Generate $T - 1$ random permutations on $d_i$ elements $\{\pi_1, \cdots, \pi_{T-1}\}$. Let $\pi_T$ be the identity permutation.
6:      // Aggregate phase
7:      For all $d \in [d_i]$, send to neighbor $d$ the ciphertext $[b_i]_{pk^{(1)}_{i\to d}}$ and the public key $pk^{(1)}_{i\to d}$.
8:      for $t = 1$ to $T - 1$ do
9:          for Neighbors $d \in [d_i]$ do
10:             Wait to receive ciphertext $c^{(t)}_{d\to i}$ and public key $k^{(t)}_{d\to i}$.
11:             Let $d' \leftarrow \pi_t(d)$.
12:             Compute $k^{(t+1)}_{i\to d'} = k^{(t)}_{d\to i} \circledast pk^{(t+1)}_{i\to d'}$.
13:             Compute $\hat{c}^{(t+1)}_{i\to d} \leftarrow \mathsf{AddLayer}\left(c^{(t)}_{d\to i}, sk^{(t+1)}_{i\to d'}\right)$ and $[b_i]_{k^{(t+1)}_{i\to d'}}$. // ciphertext under key $k^{(t+1)}_{i\to d'}$
14:             Compute $c^{(t+1)}_{i\to d'} \leftarrow \mathsf{HomOR}\left([b_i]_{k^{(t+1)}_{i\to d'}}, \hat{c}^{(t+1)}_{i\to d'}\right)$.
15:             Send $c^{(t+1)}_{i\to d'}$ and $k^{(t+1)}_{i\to d'}$ to neighbor $d'$.
16:         end for
17:     end for
18:     Wait to receive $c^{(T)}_{d\to i}$ and $k^{(T)}_{d\to i}$ from each neighbor $d \in [d_i]$.
19:     Compute $[b_i]_{k^{(T)}_{d\to i}}$ and let $e^{(T)}_{d\to i} \leftarrow \mathsf{HomOR}\left(c^{(T)}_{d\to i}, [b_i]_{k^{(T)}_{d\to i}}\right)$
20:     // Decrypt phase
21:     for $t = T$ to 1 do
22:         For each $d \in [d_i]$, send $e^{(t)}_{i\to d'}$ to $d' = \pi_t^{-1}(d)$. // Passing back
23:         for $d \in [d_i]$ do
24:             Wait to receive $e^{(t)}_{d\to i}$ from neighbor $d$.
25:             Compute $d' \leftarrow \pi_t^{-1}(d)$.
26:             $e^{(t-1)}_{i\to d'} \leftarrow \mathsf{DelLayer}\left(e^{(t)}_{d\to i}, sk^{(t)}_{i\to d'}\right)$ // If $t = 1$, DelLayer decrypts.
27:         end for
28:     end for
29:     // Produce output bit
30:     $b \leftarrow \bigvee_{d\in[d_i]} e^{(0)}_{i\to d}$.
31:     Output $b$.
32: end procedure
```

5.1 Proof of simulation security

Recall that a protocol Π is a *topology-hiding broadcast* protocol if it is a topology-hiding realization of $\mathcal{F}_{\text{Broadcast}}$. In this section, we will show that protocol 1 is a topology-hiding broadcast, 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, δ-Exploration Sequences for any $\delta = negl(\kappa)$ of length T, Protocol 1 is complete; by the end of the protocol, every node gets the output bit with all but negligible probability in the security parameter κ.*

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

So, consider a single node, u_0, with bit b_0. In the protocol, u_0's neighbors are identified by pseudonyms: u_0 just numbers them 1 to d_{u_0} and identifies them that way. We will follow one sequence that starts at u_0 with bit b_0; u_i will identify the ith node in the sequence. For the sake of notation, pk_i will denote the public key generated by node u_i at step $i+1$ for node u_{i+1} (so $pk_i = pk^{(i+1)}_{u_i \to u_{i+1}}$), and k_i will be the aggregate key-product at step i (so $k_i = pk_0 \circledast \ldots \circledast pk_i$).

- On the first step, u_0 encrypts b_0 with pk_0 into c_1 and sends it and public key pk_0 to one of its neighbors, u_1. We will follow c_1 on its walk through T nodes.
- At step $i \in [T-1]$, c_i was just sent to u_i from u_{i-1} and is encrypted under the product $k_{i-1} = pk_0 \circledast pk_1 \circledast \cdots \circledast pk_{i-1}$, also sent to u_i. u_i computes the new public key $pk_0 \circledast \cdots \circledast pk_i = k_i$, adding its own public key to the product, encrypts b_i under k_i, and re-encrypts c_i under k_i via $\mathsf{AddLayer}$. Then, using the homomorphic OR, u_i computes c_{i+1} encrypted under k_i. u_i sends c_{i+1} and k_i to $u_{i+1} = \pi_i^{(u_i)}(u_{i-1})$.
- At step T, node u_T receives c_T, which is the encryption of $b_0 \vee b_1 \vee \cdots b_{T-1}$ under key $pk_0 \circledast \cdots \circledast pk_{T-1} = k_{T-1}$. u_T encrypts and then OR's his own bit to get ciphertext $e_T = \mathsf{HomOR}(c_T, [b_T]_{k_{T-1}})$. u_T sends e_T back to u_{T-1}.
- Now, on its way back in the decrypt phase, for each step $i \in [T-1]$, u_i has just received e_i from node u_{i+1} encrypted under $pk_1 \circledast \cdots \circledast pk_i = k_i$. u_i deletes the key layer pk_i to get k_{i-1} and then using $\mathsf{DelLayer}$, removes that key from encrypting e_i to get e_{i-1}. u_i sends e_{i-1} and k_{i-1} to $u_{i-1} = (\pi_i^{(u_i)})^{-1}(u_{i+1})$.
- Finally, node u_0 receives e_0 encrypted only under public key pk_0 on step 1. u_0 deletes that layer pk_0, revealing $e_0 = b_0 \vee \cdots \vee b_T$.

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

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

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

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

Lemma 5.2. *If the underlying encryption OR-homomorhpic PKCR scheme is CPA-secure and a full collection of non-interfering, information-local, δ-Exploration Sequences for any $\delta = negl(\kappa)$ of length T, then protocol 1 realizes the functionality of $\mathcal{F}_{Broadcast}$ in a topology-hiding way against a statically corrupting, semi-honest adversary.*

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

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

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

3. $\mathcal{S}$ gets the local neighborhood for each $P \in \mathcal{Q}$: $\mathcal{S}$ knows how many neighbors each P has and if that neighbor is also in $\mathcal{Q}$, but doesn't need to know anything else about the topology [6].

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

   - **Simulating messages from honest parties in Aggregate phase.** For every $Q \in \mathcal{N}(P)$ and $Q \notin \mathcal{Q}$, $\mathcal{S}$ simulates Q as follows. At the start of the algorithm, $\mathcal{S}$ creates T key pairs:
   $$(pk^{(1)}_{Q\to P}, sk^{(1)}_{Q\to P}), \cdots, (pk^{(T)}_{Q\to P}, sk^{(T)}_{Q\to P}) \leftarrow \mathsf{Gen}(1^\kappa)$$
   At step $t = i$ in the for loop on line 8, $\mathcal{S}$ simulates Q sending a message to P by sending $([0]_{pk^{(i)}_{Q\to P}}, pk^{(i)}_{Q\to P})$. $\mathcal{S}$ receives the pair $(c^{(i)}_{P\to Q}, k^{(i)}_{P\to Q})$ from P at this step.
   - **Simulating messages from honest parties in the Decrypt phase.** Again, for every $P \in \mathcal{Q}$, $Q \in \mathcal{N}(P)$ and $Q \notin \mathcal{Q}$, $\mathcal{S}$ simulates Q. At $t = i$ in the for loop on line 20, $\mathcal{S}$ sends $[b_{out}]_{k^{(i)}_{Q\to P}}$ to P. $\mathcal{S}$ receives $e^{(i)}_{P\to Q}$ from P.

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

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

   (b) Hybrid 2. $\mathcal{S}$ replaces the real keys with simulated public keys, but still knows everything about the graph (as in Hybrid 1). Formally, for every honest Q that is a neighbor to $P \in \mathcal{Q}$, $\mathcal{S}$ generates
   $$(pk^{(1)}_{Q\to P}, sk^{(1)}_{Q\to P}), \cdots, (pk^{(T)}_{Q\to P}, sk^{(T)}_{Q\to P}) \leftarrow \mathsf{Gen}(1^\kappa)$$
   and instead of adding a layer to the encrypted $[b]_{pk^*}$ that P has at step t, as done in line 12 and 13, $\mathcal{S}$ computes $b' \leftarrow b_Q \vee b$ and sends $[b']_{pk^{(t)}_{Q\to P}}$ to P during the aggregate phase; it is the same message encrypted in Hybrid 1, but it is now encrypted under an unlayered, fresh public key. In the decrypt phase, each honest Q neighbor to P will get back the bit we get from the sequence of OR's encrypted under that new public key as well; the way all nodes in $\mathcal{Q}$ peel off layers of keys guarantees this will still be correct.

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

[6]Recall that from definition 4.1, $\mathcal{F}_{\text{graphInfo}}$ does not reveal if nodes in $\mathcal{Q}$ have neighbors in common. All $\mathcal{S}$ needs to know is which neighbors are also in $\mathcal{Q}$.

(d) 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}.

(e) Hybrid 5. $\mathcal{S}$ finally simulates the ideal functionality at the during the decrypt phase, sending fresh encryptions of b_{out}, the output of $\mathcal{F}_{\text{Broadcast}}$, under the simulated public keys. This is instead of simulating the sequences through the graph and ORing only specific bits together. Notice that this hybrid is equivalent to our original description of $\mathcal{S}$ and requires no knowledge of other parties' values or of the graph topology other than local information about Q (as specified by the $\mathcal{F}_{\text{graphInfo}}$ functionality).

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

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

(b) Now we will show that no PPT adversary can distinguish between Hybrids 2 and 3. The only difference between these two hybrids is that $\mathcal{A}$ sees encryptions of the broadcast bit as it is being transmitted as opposed to seeing only encryptions of 0 from the simulator. Note that the simulator chooses a key independent of any key chosen by parties in Q in each of the aggregate rounds, and so the bit is encrypted under a key that $\mathcal{A}$ does not know. This means that if $\mathcal{A}$ can distinguish between these two hybrids, then $\mathcal{A}$ can break semantic security of the scheme, distinguishing between encryptions of 0 and 1.

(c) Next, we will show that Hybrids 3 and 4 are statistically indistinguishable.

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

Lemma 5.1 showed that each sequence has probability at least $1 - \text{negl}(\kappa) = 1 - \delta$ of covering the graph. There are two sequences starting at each edge, making for at most $2n^2$ simultaneous sequences. So, the probability that there exists a sequence S so that $b_{out} \neq b_S$ is at most $2n^2 \cdot \text{negl}(\kappa) = \text{negl}(\kappa)$ by a union bound. Therefore, this difference in hybrids is undetectable to any polynomial adversary.

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

The difference between these hybrids is that our simulated encryption of b_{out} is generated by making a fresh encryption of b_{out}. By the claim 3.1, the encryption generated by ORing many times in the graph is computationally indistinguishable to a fresh encryption of b_{out}. Therefore, computationally, it is impossible to distinguish between Hybrids 4 and 5.

□

5.2 Statement and Proof of Main Theorem

Now that we have shown protocol 1 satisfies definition 4.1 for topology-hiding broadcast in lemma 5.2, we can formally state and prove the main theorem (that there is topology-hiding broadcast), and its corollary (that there is topology-hiding computation).

Theorem 5.3 (Topology-hiding broadcast for all network topologies)**.** *Suppose there exists an OR-homomorphic PKCR and a full collection of information-local non-interfering negl(κ)-exploration sequences of length $T = poly(\kappa)$. Then, there exists a polynomial-time protocol Π_n that is a topology-hiding broadcast over any network topology G of at most $n = poly(\kappa)$ nodes. Moreover, this protocol takes $2T$ rounds.*

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 Π is a *topology-hiding protocol for $\mathcal{F}$* if it is a polynomial-time topology-hiding realization of $\mathcal{F}$.

Corollary 5.4 (Topology-hiding computation for all network topologies)**.** *Suppose there exists an OR-homomorphic PKCR and a full collection of information-local non-interfering negl(κ)-exploration sequences of length $T = poly(\kappa)$. Then for every polynomial-time functionality $\mathcal{F}$, there exists a protocol Π_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 $\mathcal{F}$, Π_n, that only takes $2T \cdot T'$ rounds.*

Proof sketch. theorem 5.3 states that our assumptions imply topology-hiding broadcast on all network topologies in $2T$ rounds. First, we can get oblivious transfer from (OR-)homomorphic encryption, [23], and from there we can get MPC: with the existence of oblivious transfer, there exist efficient MPC protocols for every efficiently computable functionality [29, 13, 12]. Each round in the MPC protocol becomes a single multi-bit broadcast; since our topology-hiding broadcast protocol is secure in the UC model, we can have each party broadcasting many messages at once (or broadcasting multi-bit messages). Since each broadcast takes $2T$ rounds, the total number of rounds is $2T \cdot T$. □

Proof of theorem 5.3. We will show that protocol 1 is the topology-hiding realization of $\mathcal{F}_{\text{Broadcast}}$. Since we assume existence of an OR-homomorphic PKCR, we are able to run our protocol. The rest of this proof is simply combining the results of lemma 5.1 and lemma 5.2.

To show protocol 1 is complete, lemma 5.1 states that for our parameter κ, protocol 1 outputs the correct bit for every node with probability at least $1 - \text{negl}(\kappa)$. This means, our protocol is correct with overwhelming probability with respect to the security parameter κ.

To show our protocol is sound, lemma 5.2 states that for our input parameter κ, an adversary can distinguish a simulated transcript from a real transcript with probability negligible in κ. Therefore, protocol 1 is sound against all PPT adversaries: they have only a negligible chance with respect to κ of distinguishing the simulation versus a real instantiation of the protocol. □

In sections section 3.7.1 and 3.7.2, we demonstrate two ways that we can construct a non-interfering, full collection of δ-exploration sequences (one with negligible δ and the other with $\delta = 0$): correlated random walks, and UESs respectively. So, we get the following two corollaries for free from the main theorem.

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

Corollary 5.5. *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 3.14 shows that with a collection of correlated random walks of length $\kappa \cdot 8n^3$, we get $2^{-\kappa}$-exploration sequences. By theorem 5.3, we get a $2 \cdot \kappa \cdot 8n^3$-round topology-hiding broadcast from protocol 1 (we have to go forward through the walk and then back, hence the extra factor of 2). □

There are two sources of error in the construction in corollary 5.5: OR'ing Homomorphically using our ElGamal construction may fail with negligible (but non-zero) probability, and some of the random walks could fail to cover the entire graph. We can get rid of this second source of error if instead of using random walks, we use UESs. So, if we had a perfectly-complete OR-Homomorphic PKCR, then we could get a perfectly-complete topology-hiding broadcast protocol.

Corollary 5.6. *If there exists an OR-Homomorphic PKCR without (even negligible) error, then there exists a polynomial-round topology-hiding broadcast for any graph G that always succeeds.*

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

Now let us analyze each walk. Every walk hits every single node in the graph because it follows a UES. So, the bit produced by every walk is going to be the OR of every node's bit, including the broadcaster's. Since there is no error in OR'ing bits together, this is guaranteed from lemma 5.1. So, every walk results in the output bit, and hence every party gets the output bit, so protocol 1 is perfectly complete. □

5.3 Complexity and Optimizations

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

5.3.1 Communication Complexity with Correlated Random Walks

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

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

Corollary 5.8. *On input n, κ, 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 3.12), 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: $\Theta(B\kappa m) = \Theta(\kappa \cdot n^3 \cdot n^2) = \Theta(\kappa n^5)$. □

Type of Graph	Cover time
Arbitrary graph [24]	$O(n^3)$
Expanders [5]	$O(n \log n)$
Regular Graphs [20]	$O(n^2)$

Table 2: Cover times for specific graphs.

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

Cover time has been studied for various kinds of graphs, and so if we leak the kind of graph we are in (e.g. expanders), then we can use a better upper bound on the cover time, shown in table 2.

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 $O(n^3)$ [5]. This means that for expanders, we can run in $C_G = O(n \log n)$ round complexity, and $O(C_G \kappa m) = O(\kappa m n \log n)$ communication complexity. Even assigning the worst case bound $m \leq n^2$, we get round and communication complexity $O(n \log n)$ and $O(\kappa n^3 \log n)$ respectively—much better than the general case that has $O(\kappa n^3)$ round complexity and $O(\kappa n^5)$ communication complexity.

5.3.2 Communication Complexity with Universal Exploration Sequences

Unfortunately, we are less precise when discussing communication complexity of our protocol when using UESs. This is because known explicit, deterministic, polynomial-time constructions use *log-space*, and these works, as far as we could find, do not discuss how long the resulting sequence is. Moreover, every source with the exception of Koucky's thesis only discusses d-regular graphs[21]. 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.[7] 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 5.9 ([28]). *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 [21].

Theorem 5.10 ([21]). *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 5.11. *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.*

[7]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

Proof. We will essentially be applying theorem 5.9 in conjunction with theorem 5.10 $O(n^2)$ 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 $O(n^2T)$. □

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

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 [25, 18, 1]. It leaves open the feasibility question against a malicious or adaptive adversary.

Although there are impossibility results for even very weak malicious adversaries (fail-stop) [25], the adversary is able to learn about the topology of the graph because it is able to disconnect it. So, if we first limit the adversary so that it cannot disconnect the graph, there is hope that we can get some results (say a t-connected graph and the adversary can abort/control at most t nodes). We can also consider models that allow some bounded leakage, so that even if an adversary disconnects the graph, she only learns a small amount about the topology.

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.

Finally, there is the question of what other cryptosystems are OR-homomorphic PKCR encryption schemes. Importantly, lattice-based cryptography may also be viable, since it has homomorphic properties and structure. Constructing topology-hiding computation from other assumptions would strengthen the result, and in the case of lattice-based cryptography, would potentially give us post-quantum security.

Topology hiding computation is a relatively unexplored subject in cryptography, having (in the computational setting) its first feasibility result in 2015 [25]. It will be exciting to see what else can be proved in this model.

References

[1] A. Akavia and T. Moran. Topology hiding computation beyond logarithmic diametere. In *To appear in Eurocrypt*, 2017.

[2] 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.

[3] 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.

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

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

[13] 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.

[14] 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.

[15] 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.

[16] S. Halevi, Y. Lindell, and B. Pinkas. Secure computation on the web: Computing without simultaneous interaction. In 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*, pages 132–150. Springer, 2011.

[17] 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).

[18] 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.

[19] 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.

[20] 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.

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

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

[23] B. Malek and A. Miri. Optimal secure data retrieval using an oblivious transfer scheme. In *WiMob'2005), IEEE International Conference on Wireless And Mobile Computing, Networking And Communications, 2005.*, volume 2, pages 25–31 Vol. 2, Aug 2005.

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

[25] 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.

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

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

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

[29] 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 [22]. Specifically, we will be proving that the algorithms we referenced in section 3.6, re-randomization and proxy-reencryption, are correct and sound. This will finish the proofs in section 3.6, showing that Cocks' public-key encryption scheme is XOR-homomorphic PKCR.

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

Most of this new notation and helper lemmas are only used to prove that we can re-randomize Cocks' ciphertexts. Once you have that, homomorphic XOR and proxy re-encryption directly follow.

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 3.6 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 [7]. 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 γ^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 γ^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, $ax+b = 1\cdot(c_p x+d_p)^2 \mod p$ and $ax+b = u_q(c_q x+d_q)^2 \mod q$. By the Chinese Remainder Theorem, we can find a $t \in \mathbb{Z}_N$ so that $t \equiv 1 \mod p$ and $t \equiv 1/u_q \mod q$, as well as $a' \equiv c_p \mod p$ and $c_q \mod q$ and $b' \equiv d_p \mod p$ and $d_q \mod q$. 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 [7]). 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 ([22]). 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

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

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
```

Claim A.5 ([22]). 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

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

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

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

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

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

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 $t \in \mathbb{J}_1$:

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

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

Now, given all of our machinery, it is easy to see how we get homomorphic addition: we multiply two ciphertexts together in $\mathcal{R}_N$, and then re-randomize it to get a random Cocks' ciphertext.

```
function QR.XOR-Hom($c_1, c_2, R$)
    Compute $ax+b=(2x+c_1)(2x+c_2)^2 \pmod{x^2-R}$
    $c_3$ ← QR.Convert($a, b, R$)
    return QR.Rand($c_3, R$)
end function
```

A.3 Proxy-Reencryption for Cocks' Ciphertexts

For details on the definition of proxy-reencryption, we direct the reader to [22]. For our purposes of this paper, we only need that if $T=r/r'$, then QR.ReEncrypt(c, R, R', T) outputs c', a Cocks' ciphertext under public key R' decrypting to the same message as c, which was under public key R.

```
function QR.ReEncrypt($c, R, R', T$)
    Compute $ax+b=2Tx+c_1$
    $c_3$ ← QR.Convert($a, b, R'$)
    return QR.Rand($c_3, R'$)
end function
```

Theorem A.6 ([22])**.** QR.ReEncrypt *is an efficient algorithm such that when given a Cocks public key $pk=(N,R=r^2)$, ciphertext $[m]_R$, and re-encryption key $T=r/r'$, outputs a ciphertext $c'=[m]_{R'}$ of the same message encrypted under public key $R'=r'^2$, computationally indistinguishable from a fresh encryption of m under R':*

$$\{\mathsf{QR.ReEncrypt}(N,R,R',c,T),r'\}\to c'\equiv_c\{\mathsf{QR.Enc}(N,R',m),r'\}$$

Proof. Given the machinery we already have (namely that Convert and Rand work correctly), we will simply prove that $ax + b$ is a linear-function ciphertext under public key R', decrypting to the same message as $2x + c$ under public key R:

$$\left(\frac{ar' + b}{N}\right) = \left(\frac{2Tr' + c}{N}\right) = \left(\frac{2\frac{r}{r'} \cdot r' + c}{N}\right) = \left(\frac{2r + c}{N}\right) = (-1)^m$$

Since $ax + b$ decrypts correctly to m with r', convert gives us a valid Cocks' ciphertext under R', which will also decrypt correctly. Since QR.Rand statistically re-randomizes the ciphertext, we return a c_3' that is indistinguishable from a fresh encryption of m under public key R'. □