Relationships Between Functionality, Security, and Privacy

by

Rio LaVigne

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author .....
Department of Electrical Engineering and Computer Science
February 6, 2020

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

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

Relationships Between Functionality, Security, and Privacy

by

Rio LaVigne

Submitted to the Department of Electrical Engineering and Computer Science
on February 6, 2020, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

One of the core goals of cryptography is to be able to offer security and privacy without sacrificing functionality. This thesis describes three different functionalities and notions of security and privacy and how they interplay. First, we delve into Topology-Hiding Computation, which has a very restrictive definition of privacy making it difficult to construct; nevertheless we provide a two constructions. Second, we invent a new cryptographic concept called Property Preserving Hashes, where we want to compress inputs while securely preserving a certain property between them, e.g. hamming distance. Finally, we explore Fine-Grained Cryptography, and develop a public key cryptosystem. In this notion of cryptography, security takes on a much less restrictive role (e.g. adversaries must run in $O(n^{100})$ time), but the protocols and security reductions must run in "fine-grained" time, e.g. less than $O(n^2)$.

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

Acknowledgments

This is the acknowledgements section. You should replace this with your own acknowledgements. TODO.

I would first like to thank my advisors Vinod Vaikuntanathan and Virginia Vassilesvka Williams. A special thank you to Vinod for supporting me during my entire time at MIT, ...

Tal Moran

Elette Boyle

Andrea Lincoln

Other coauthors

Dan Boneh

Other members of the theory community.

Other members of the crypto community.

Family and Avery <3

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the core goals of cryptography is to be able to offer security and privacy without sacrificing functionality. To do this, cryptographers define notions of both correctness and security. We can then build systems that we prove satisfy both of these definitions. Correctness states that the system provides the functionality we want, while security is essentially the notion that we can effectively hide information from an adversary. Depending on the notion of security, this adversary may be all powerful (information-theoretic or computationally unbounded), may run only in polynomial time (computationally bounded), or even bounded by a specific polynomial runtime like $O(n^2)$ (a fine-grained adversary). This thesis will focus on realizing three different functionalities, each one hiding different information, and three different notions of security.

In the first part of this thesis, we discuss results in the area of Topology Hiding Computation (THC). THC is a generalization of secure multiparty computation. In this model, parties are in an incomplete but connected communication network, each party with its own private inputs. The functionality these parties want to realize is evaluating some function (computation) over all of their inputs. On the security and privacy side, these parties want to hide both their private inputs and who their neighbors are. So, the goal is for these parties to run a protocol, communicating only with their neighbors, so that by the end of the protocol, they learn the output of the function on their inputs and *nothing else*, including information about what the communication network looks like other than their own neighborhood.

It turns out that THC is a difficult notion to realize, mired in impossibility results. We circumvent two impossibility results in THC in two different ways. First, there is an impossibility result for THC stating that it is impossible to design THC against adversaries that can "turn off" parties during the protocol [107]. In essence, an adversary that has this power can always learn something about the graph. So, instead of designing a protocol that would attempt to leak no information, we designed a protocol to limit the amount of information leaked as much as possible. We ran into similar impossibility results when considering a model more similar to the real-world: channels between parties now have unknown delays on them, and an adversary may be able to control the delay. We had to find a model that mirrored this real-world complication, and this was also not impossible to achieve.

In the second part of the thesis, we focus on Adversarially Robust Property Preserving Hashes (PPH). PPHs are a generalization of collision resistant hash functions (CRHFs). Recall that a CRHF is a family of hash functions $\mathcal{H} = \{h : \{0,1\}^n \to \{0,1\}^m\}$ that is compressing ($m < n$), and has the property that no Probabilistic Polynomial-Time (PPT) adversary given h can find two inputs $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$ except with negligible probability. Looking at CRHFs from a slightly different perspective, their functionality is to preserve the equality predicate between two inputs while compressing them, preventing an adversary from coming up with inputs where the equality predicate is not preserved (even though these inputs exist, they are hard to find). A PPH is also a compressing function that preserves some other property. For example, we focus on the property of "gap-hamming" distance: if two inputs are close in hamming distance, then their hashes are also close, and if two inputs are far, their hashes are also far. Note that this example is of a promise predicate since we do not care about inputs that are neither close nor far.

Property preserving hashes were a new cryptographic primitive that we designed, and so it was important to understand what was possible and what was not in our new model. We found strong connections between One-Way Communication Complexity (OWC complexity) and our primitive. Fortunately for us, OWC is a rich, well-studied area of complexity. In many cases, we could not directly port OWC results to our setting, but we could use their proof techniques and get lower bounds. Once we had these lower bounds, we had a much better sense of what was and was not possible, and could construct PPHs more easily.

In the final chapter, for a different perspective, we design a public-key cryptography functionality against weak adversaries. This functionality allows for a party to publish a public key while they keep the secret key, and any other party to use that public key to encrypt a message that only the party with the secret key can decrypt, *hiding* that message from any eavesdroppers. Unlike in standard models for cryptography, however, our eavesdroppers much less powerful: their runtime is bounded by an explicit polynomial instead of "probabilistic polynomial-time", e.g. can only run in time $O(n^{100})$. This, of course, is only interesting as a cryptographic notion if the adversary has the same or more time to run than the honest parties. So, we get a notion of cryptography we call *fine-grained*: for examples, honest parties might only need to run in $O(n^2)$ time, while any adversary running in time $O(n^{100})$ time still cannot glean any useful information with some probability. Motivating the study of this cryptography is the fact that it does not rely on any of the normal cryptographic assumptions, including $P \neq NP$, $BPP \neq NP$, or even the assumption that one-way functions exist.

However, due to its fine-grained nature many standard cryptographic reductions (like Goldreich-Levin hardcore bits [64]) no longer work. And so, in this thesis we demonstrate variations that can work in our setting, including a notion of fine-grained one-way functions, fine-grained hardcore-bits, a fine-grained key exchange, and finally fine-grained public-key cryptography.

1.1 Results

In this thesis, we explore these three different notions of hiding while preserving a functionality:

1. hide private inputs and network topology while preserving computation,
2. hide as much information as possible while preserving a property between inputs,
3. and hide messages from a weak adversary while preserving the communication and fine-grained runtime.

Topology-Hiding Computation To address the first perspective, we explore the realm of topology-hiding computation (THC). THC is a generalization of secure multiparty computation (MPC), where we hide not only each party's input, but also the communication graph; parties know who their neighbors are, but learn nothing else about the structure of the graph. In 2017, we showed that THC is possible against a very weak adversary [4], but there were still many open questions surrounding the nature of THC.

- **Fail-stop Adversaries.** Fail-stop adversaries can turn off parties during the protocol. As shown by Moran, Orlav, and Richelson in 2015 [107], it is impossible to achieve perfect topology hiding when giving the adversary this power. The next question is how to get around such an impossibility while still providing meaningful privacy and security. Since the adversary was going to learn something in any protocol we could devise, we decided to use a model that would incorporate this leakage.

  Our results here were a definition of security that included access to a "leakage oracle" and a protocol that used this oracle in the security proof. This oracle would spit out yes or no answers to specific queries, leaking one bit of information at a time. In the security reduction of our protocol, a simulator would only need to call this oracle at most once, leaking at most one bit of information to an adversary. Concretely, we could bound the probability that the simulator would have to call the leakage oracle, meaning that we would leak only a polynomial-fraction of a bit. Our protocol ran in $\tilde{O}(\kappa n^4/\rho)$ rounds, where κ is the security parameter, n is the number of parties, and ρ is the probability that we leak a bit.

  These results are detailed in our paper at TCC 2018 [94].

- **Probabilistic Delay Model.** This model was the result of first trying to build THC in a purely asynchronous setting, only to prove that it was impossible to do so. So, to get around this without just reducing ourselves to the fail-stop case, we designed a new model that more closely resembled an asynchronous, real-world setting.

More formally, each edge between parties in the Probabilistic Delay Model has its own distribution over delays. Each time a party sends a message to another party, that delay is sampled *independently*. This simulates real world traffic having different delays as it gets sent from one server to another.

Our results also included two new constructions in this model. The first is a construction that works under standard cryptographic assumptions, but only on trees, cycles, or graphs with small circumference. The second is a construction that requires secure hardware boxes/tokens, but works for arbitrary communication graphs. Although this is a very powerful assumption, there are still many challenges involved since an adversarial party can query this hardware at any point with any input.

These results are currently in submission.

Adversarially Robust Property Preserving Hashes and One-Way Communication Lower Bounds Addressing the second perspective, we look at a new cryptographic primitive: adversarially robust Property Preserving Hashes (PPH). The inspiration here is two-fold. First, collision-resistant hash functions (CRHFs) are a staple in cryptography: no PPT adversary can produce an $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$ except with negligible probability over h sampled from the CRHF family and coins of the adversary, and hence one can preserve the "equality" predicate on compressed inputs even against adversarially chosen inputs. We want to compress inputs while maintaining some property other than equality between them, even in the presence of adversarially chosen inputs. The properties we considered were those that already had non-robust constructions, in particular locality-sensitive hashing (LSH) [80]. An LSH family is a hash function family $\mathcal{H} = \{h : \{0,1\}^n \to \{0,1\}^m\}$ that compresses inputs ($m < n$) and is *mostly* correct. For example, say we were preserving a gap-ℓ_p norm for a gap between r and cr for some constant $c > 1$ (note that in our case, gap-hamming is equivalent to gap-ℓ_0 norm). Then, we have a threshold τ so that for any pair $x_1, x_2 \in \{0,1\}$

- if $||x_1 - x_2||_p < r$, $||h(x_1) - h(x_2)||_p < \tau$, and
- if $||x_1 - x_2||_p > cr$ then $||h(x_1) - h(x_2)||_p \geq \tau$

with high probability over our choice of h sampled from $\mathcal{H}$.

We can use almost the same language to define PPHs for some property $P : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, except our probability will be negligible and taken over both our sampled hash function and the coins of a PPT adversary. We also generalize the predicate evaluation: instead of using the same predicate on hashed values, we can use an "evaluation function" called Eval. So, even in the presence of (PPT) adversarially chosen values of x_1 and x_2,

- if $P(x_1, x_2) = 0$, then $\mathsf{Eval}(h(x_1), h(x_2)) = 0$, and
- if $P(x_1, x_2) = 1$, then $\mathsf{Eval}(h(x_1), h(x_2)) = 1$

with all but negligible probability. In other words, for any PPT adversary given h sampled from $\mathcal{H}$, the probability that the adversary produces x_1 and x_2 such that $P(x_1, x_2) \neq \mathsf{Eval}(h(x_1), h(x_2))$ is negligible.

The link between robust PPH and LSH was very clear, but less obvious was the link between PPH and one-way communication (OWC): in essence, compressing an input as much as possible for a OWC protocol is akin to compressing an input as much as possible while preserving some property of that input as we would want to do for a hash function. These connections led to the following results, which will be included in the thesis.

- OWC is a rich field with a lot of work detailing lower bounds on the complexity of certain predicates. These lower bounds *almost* directly translated to impossibility results for PPHs. However, we needed to be careful because OWC lower bounds dealt with constant error, where as cryptographers, we care about negligible error. We characterized these OWC lower bounds and how they mapped to PPH lower bounds. We showed that a class of predicates we call "reconstructing predicates" could not have PPHs. This class includes useful predicates like GreaterThan, Index, and ExactHamming.

- With these lower bounds and previous results from LSH in mind, we turned to constructing a PPH for Gap-Hamming: the promise predicate where we can distinguish between pairs of points that are $(1-\epsilon)d$ *close* in hamming distance or $(1+\epsilon)d$ *far*. For all pairs within the gap, we "don't care" how our PPH behaves. We build two constructions for this predicate, each with different drawbacks and benefits. Our first construction relies only on collision-resistant hashing. Our second uses a new assumption related to the hardness of syndrome decoding [10].

- Rounding out this paper and its myriad of definitions, we demonstrate that even for very simple predicates, we require collision resistance, and even if we weaken our main definition of a robust PPH, we still need one-way functions.

These results were published in ITCS 2019 [29].

In unpublished follow-up research with Cyrus Rashtchian, we explored the intricacies of the CRHF method for constructing gap-hamming PPHs. We fleshed out the relationship between d, the "center" of our gap, and the size of the hash function output. We also discussed how to use this kind of construction to get a gap-ℓ_1-norm PPH and found that with standard methods of transforming ℓ_1 into ℓ_0, it could not work.

Fine-Grained Cryptography and Barriers Addressing the final perspective, this thesis will discuss our work in Fine-Grained cryptography. With my coauthors Lincoln and Vassilevska Williams, we built the first fine-grained key exchange built from fine-grained assumptions. The premise was to explore what cryptography could be like in "Pessiland," a world in which there are no cryptographic one-way functions (which also implies no public key cryptography). We looked at this through the lens of

fine-grained complexity, using both assumptions and reduction techniques from that field.

While designing cryptography for this setting, we came across multiple barriers. The first was that some fine-grained problems would not lend themselves to build cryptography without refuting NSETH [35]. So, we would need to use a fine-grained problem that was not associated with that barrier. Another form of barrier we ran into was worst-case to average-case reductions for problems that would make for good cryptography. Even now, the only worst-case to average-case reductions for fine-grained problems are for counting versions of these problems. Unfortunately, counting-style problems do not lend themselves well to building cryptography, as there are no longer small enough witnesses. Finally, many standard cryptographic reductions no longer work in a fine-grained world, since they take a non-trivial polynomial amount of time, and so a challenge we faced was to ensure all of our reductions were fine-grained and build different types of primitives based off of these restricted reductions.

Despite these difficulties, we achieved the following results:

- Assuming that an average-case version of Zero-k-Clique requires $n^{k-o(1)}$ time, we constructed a non-interactive key exchange that required honest parties to run in $O(N)$ time and an adversary to run in time $\tilde{\Omega}(N^{1.5-\epsilon})$, where ϵ decreases with respect to k.[1]
- Generalize the properties of Zero-k-Clique to construct this key exchange: plantable, list-hard, and splittable.
- Define and construct fine-grained hard-core bits.

This work was published in CRYPTO 2019 [93].

In unpublished follow-up work, we also show how to use another property of Zero-k-Clique to construct a key exchange where the adversary now needs $\Omega(N^{2-\epsilon})$ time. This N^2 gap is the best we are able to do since we base these constructions on Merkle puzzles [17].

1.2 Related Works

1.2.1 Related Work to Topology Hiding Computation

1.2.2 Related Work to Property Preserving Hashing

1.2.3 Related Work to Fine-Grained Cryptography

1.3 Outline of Thesis

TODO

[1] The tilde in $\tilde{\Omega}$ ignores any *subpolynomial* factors.

Chapter 2

Topology Hiding Computation

2.1 Overview

This chapter is based on [94] and [95].

Secure communication over an insecure network is one of the fundamental goals of cryptography. A fundamental solution to this problem is secure multiparty computation. Here, one commonly assumes that all parties have pairwise communication channels. In contrast, for many real-world scenarios, the communication network is not complete, and parties can only communicate with a subset of other parties. A natural question is whether a set of parties can successfully perform a joint computation over an incomplete communication network while revealing no information about the network topology.

The problem of *topology-hiding computation* (THC) was introduced by Moran et al. [108], who showed that THC is possible in the setting with passive corruptions and graphs with logarithmic diameter. Further solutions improve the communication efficiency [74], and allowed for larger classes of graphs [6, 4]. A natural next step is to extend these results to settings with more powerful adversaries. Unfortunately, even a protocol in the setting with fail-corruptions (in addition to passive corruptions) must leak topological information about the graph [108].

A comparison of our results to previous works in topology-hiding communication is found in Tables 2.1 and 2.2.

Privately Key-Commutative Randomizeable Encryption (PKCR). All of the standard-model results (i.e. based on standard cryptographic assumptions) use a special form of encryption called OR-Homomorphic PKCR. This kind introduced in [6] and used the Decisional Diffie-Hellman (DDH) assumption. Later in [92], it was shown you could make PKCR with the QR assumption. Here, we will show that OR-Homomorphic PKCR is also possible under the Learning-With-Errors (LWE) assumption or assuming that 'deep' Fully-Homomorphic Encryption (FHE) exists.

Along similar lines, a slightly modified form of PKCR is used in some of the results, called PKCR*. The difference between these schemes is that Fortunately, we can show that any PKCR encryption scheme can be compiled into a PKCR* scheme.

This is detailed in Section **??**.

Table 2.1: Adversarial model and security assumptions of existing topology-hiding broadcast protocols. The table also shows the class of graphs for which the protocols have polynomial communication complexity in the security parameter and the number of parties.

Adversary	Graph	Hardness Asm.	Model	Reference
semi-honest	log diam.	Trapdoor Perm.	Standard	[108]
	log diam.	DDH	Standard	[75]
	cycles, trees, log circum.	DDH	Standard	[6]
	arbitrary	DDH or QR	Standard	[3] and [92]
fail-stop	arbitrary	OWF	Trusted Hardware	[14]
semi-malicious & fail-stop	arbitrary	DDH or QR or LWE	Standard	[This Thesis] [94]
Probabilistic Unknown Delays	cycles, trees, log circum.	DDH or QR or LWE	Standard	[This Thesis] [95]
	arbitrary	OWF	Trusted Hardware	[This Thesis] [95]

Fail-stop adversaries. The first paper tackling this hurdle was [14], which provides THC for fail-stop or semi-malicious adversaries using secure hardware with pre-shared secret keys embedded in the hardware, and formally leaks a bit of information about the graph in the event of an adversarial fail-stop. In this thesis, we show a stronger result: we can construct THC protocols for fail-stop or semi-malicious adversaries under *standard* cryptographic assumptions, which still leaks at most one bit of information in the event of a fail-stop. This is covered in Section **??**.

Theorem 1 (informal)**.** *If DDH, QR or LWE is hard, then for any $p \in (0, 1]$, there exists a topology-hiding broadcast protocol for any network graph G leaking at most a fraction p of a bit, secure against an adversary that does any number of static passive corruptions and adaptive crashes. The round and communication complexity is polynomial in κ and $1/p$.*

Then, because we can compile broadcast to full computation (with the same leakage), we get the following corollary. Note that in this case, the compilation must be sequential and any abort will abort the rest of the protocol.

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

Finally, we show how to compile any protocol that tolerates a fail-stop adversary into a protocol that also tolerates semi-malicious corruption. As a corollary, we

Table 2.2: The table provides more details on the communication and round complexity of the protocols in Table 2.1. We denote by κ the security parameter, n is the number of parties, D is a bound on the diameter of the graph and d is a bound on the maximum degree of the graph. Note that for protocols in the Probabilistic Unbounded Delay Model, communication is dependent on the delays, and so it does not make sense to compare those results here.

Adversary	Communication	Rounds	Reference
semi-honest	$O(poly(d)^D n\kappa)$	$O(poly(d)^D)$	[108]
	$O((d+1)^D n\kappa)$	$5 \cdot D$	[75]
	$O(n^2\kappa)$	$O(n)$	[6]
	$O(n^5(\kappa + \log(n)))$	$O(n^3(\kappa + \log(n)))$	[5]
fail-stop	$O(nD(d+\kappa))$	$O(nD)$	[14]
semi-malicious & fail-stop	$O(n^6(\kappa + \log(n)))$	$O(n^4(\kappa + \log(n)))$	[This Thesis] [94]

obtain that any MPC functionality can be realized with arbitrarily small leakage and tolerating any number of static semi-malicious corruptions and adaptive crashes.

Theorem 2 (informal)**.** *Let $\mathcal{F}$ be an MPC functionality and let Π be a protocol that topology-hidingly realizes $\mathcal{F}$ in the presence of static passive corruptions and adaptive crashes. Then, Π can be compiled into a protocol Π' that topology-hidingly realizes $\mathcal{F}$ in the presence of static semi-malicious corruption and adaptive crashes.*

Asynchronous settings. All these prior results consider the *fully synchronous* model, where a protocol proceeds in rounds. This model makes two assumptions: first, the parties have access to synchronized clocks, and second, every message is guaranteed to be delivered within one round. While the first assumption is reasonable in practice, as nowadays computers usually stay synchronized with milliseconds of variation, the second assumption makes protocols inherently impractical because the running time of a protocol is always counted in the number of rounds, and the round length must be chosen based on the most pessimistic bound on the message delivery time.

A first attempt would be to develop a protocol for the fully asynchronous model, where the adversary has complete control over delays. Unfortunately, we can show that *any* setting where the adversary can control delays in an unbounded way, leaks information (see Section 2.10).

So, we develop a new Probabilistic Unknown Delay Model and we formally define it in Section 2.9. In this model the messages are delayed independently of the adversary, but different connections have different, unbounded probabilistic delays. This means that we do not use the assumption that makes the synchronous protocols impractical, but keep the more reasonable assumption that parties have synchronized clocks.

2.2 Preliminaries for Topology Hiding Computation

2.2.1 Graphs and Random Walks

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

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

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

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

The techniques used in the failstop protocol are based on the random-walk Topology-Hiding Broadcast protocol from [**?**].

Techniques for the delay-model protocols have to differ, since simulating delay in a random walk does not seem feasible. And because of delays, it can be difficult to know what "layer" an encryption is on — one party may be receiving messages much more frequently or much earlier than another. Our innovations are two-fold. First, we modify PKCR to PKCR*, in which no level information is given unless the message is completely decrypted. In other words. the message space and cyphertext space do not intersect. Second, we develop techniques that are based on the cycle/tree protocols from [6] for cycles and trees, and for general graphs, we use secure hardware as in [14].

2.2.2 Random Walk Protocol from [3]

Our protocols are based on the random walk protocol from [3]. We therefore give a high level overview of that protocol and explain why it is correct and sound. We recall that the random walk protocol achieves security against static passive corruptions. To achieve broadcast, the protocol actually computes an OR. Every party has an input bit: the sender inputs the broadcast bit and all other parties use 0 as input bit. Computing the OR of all those bits is thus equivalent to broadcasting the sender's message.

First, we will explain a simplified version of the protocol that is unfortunately not sound, but this gets the principal across. Each node will take its bit, encrypt it under a public key and forward it to a random neighbor. The neighbor OR's its own bit, adds a fresh public key layer, and it randomly chooses the next step in the walk that the message takes, choosing a random neighbor to forward the bit. Eventually, after about $O(\kappa n^3)$ steps, the random walk of every message will visit every node

in the graph, and therefore, every message will contain the OR of all bits in the network. Now we start the backwards phase, reversing the walk and peeling off layers of encryption.

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

In more detail, and to demonstrate security, consider a single node v with d neighbors. During the forward phase at step t, v gets d incoming messages — one from each of its neighbors — homomorphically OR's its bit to each, computes d fresh public keys, adding a layer to each, and finally computes a random permutation π_t on its neighbors, forwarding the message it got from neighbor i to neighbor $\pi_t(i)$ and so on. During the backwards phase, node v removes the public key layers it added during the corresponding forward round, and then reverses the permutation, sending the message it got from neighbor j to neighbor $\pi_t^{-1}(j)$. Because all messages are encrypted under semantically secure encryption, v cannot tell whether it has received a 0 or 1 from any of its neighbors, and because all of its neighbors are layering their own fresh public keys onto the messages, there is no way for v to tell where that message came from or if it had seen it before. Intuitively, this gives us soundness (see [3] for details).

Now, this protocol is also correct: every walk will, with all but negligible probability, visit every node in the network, and therefore every message will, with all but negligible probability, contain an encryption of the OR of all bits in the graph by the end of the forward phase. The backward phase then takes that message at the end of the walk, and reverses the walk exactly, popping off the public key layer that was added at each step. By the end of the backward phase, the node that started the walk gets the decryption of the message: the OR of all bits in the graph. Because all of the walks succeed, and every node started a walk, every node gets the correct output bit as desired.

2.2.3 OR-Homomorphic PKCR Encryption Scheme

As in [3], our protocols require a public key encryption scheme with additional properties, called *Privately Key Commutative and Rerandomizable encryption*. We assume that the message space is bits. Then, a PKCR encryption scheme should be: (1) privately key commutative and (2) homomorphic with respect to the OR operation. We formally define these properties below. [1]

[1]PKCR encryption was introduced in [6, 3], where it had three additional properties: key commutativity, homomorphism and rerandomization, hence, it was called Privately Key Commutative and *Rerandomizable* encryption. However, rerandomization is actually implied by the strengthened notion of homomorphism. Therefore, we decided to not include the property, but keep the name.

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

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

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

Privately Key-Commutative

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

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

- For every public key pair $\mathtt{pk}_1, \mathtt{pk}_2 \in \mathcal{PK}$ with corresponding secret keys $\mathtt{sk}_1$ and $\mathtt{sk}_2$, message $m \in \mathcal{M}$, and ciphertext $c = [m]_{\mathtt{pk}_1}$,

$$\mathsf{AddLayer}(c, \mathtt{sk}_2) = [m]_{\mathtt{pk}_1 \circledast \mathtt{pk}_2}.$$

- For every public key pair $\mathtt{pk}_1, \mathtt{pk}_2 \in \mathcal{PK}$ with corresponding secret keys $\mathtt{sk}_1$ and $\mathtt{sk}_2$, message $m \in \mathcal{M}$, and ciphertext $c = [m]_{\mathtt{pk}_1}$,

$$\mathsf{DelLayer}(c, \mathtt{sk}_2) = [m]_{\mathtt{pk}_1 \circledast \mathtt{pk}_2^{-1}}.$$

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

OR-Homomorphic

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

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

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

In [75], [6] and [3], the authors discuss how to get this kind of homomorphic OR under the DDH assumption, and later [5] show how to get it with the QR assumption. For more details on other kinds of homomorphic cryptosystems that can be compiled into OR-homomorphic cryptosystems, see [5].

In this paper we show how to instantiate a PKCR encryption scheme under the LWE assumption (for details, see Section 2.3.1).

2.2.4 Techniques to deal with Delay

PKCR*

Akavia-Moran Cycle Protocol [6]

Secure Hardware

2.2.5 Techniques

Blurb about PKCR techniques

Blurb about techniques from ALM and BBMM for FS.

Blurb about techniques from AM and BBMM for Delays.

2.3 Privately Key-Commutative Randomizable Encryption (PKCR)

2.3.1 LWE based OR-Homomorphic PKCR Encryption

In this section we show how to get a PKCR encryption scheme from the LWE assumption. Basis of our PKCR scheme is the public-key crypto-system proposed in [119]. Let us briefly recall the public-key crypto-system:

LWE PKE scheme [119] Let κ be the security parameter of the cryptosystem. The cryptosystem is parameterized by two integers m, q and a probability distribution χ on $\mathbb{Z}_q$. To guarantee security and correctness of the encryption scheme, one can choose $q \geq 2$ to be some prime number between κ^2 and $2\kappa^2$, and let $m = (1+\epsilon)(\kappa+1)\log q$ for some arbitrary constant $\epsilon > 0$. The distribution χ is a discrete gaussian distribution with standard deviation $\alpha(\kappa) := \frac{1}{\sqrt{\kappa} log^2 \kappa}$.

Key Generation: *Setup*: For $i = 1, \ldots, m$, choose m vectors $\mathbf{a}_1, \ldots, \mathbf{a}_m \in \mathbb{Z}_q^\kappa$ independently from the uniform distribution. Let us denote $A \in \mathbb{Z}_q^{m \times \kappa}$ the matrix that contains the vectors $\mathbf{a}_i$ as rows.

Secret Key: Choose $\mathbf{s} \in \mathbb{Z}_q^\kappa$ uniformly at random. The secret key is $\mathtt{sk} = \mathbf{s}$.

Public Key: Choose the error coefficients $e_1, \ldots, e_m \in \mathbb{Z}_q$ independently according to χ. The public key is given by the vectors $b_i = \langle \mathbf{a}_i, \mathtt{sk} \rangle + e_i$. In matrix notation, $\mathtt{pk} = A \cdot \mathtt{sk} + \mathbf{e}$.

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

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

To extend this scheme to a PKCR scheme, we need to provide algorithms to rerandomize ciphertexts, to add and remove layers of encryption, and to homomorphically compute the OR. To obtain the OR-homomorphic property, it is enough to provide a XOR-Homomorphic PKCR encryption scheme, as was shown in [3].

Extension to PKCR We now extend the above PKE scheme to satisfy the requirements of PKCR (cf. Section 2.2.3). For this we show how to rerandomize ciphertexts, how add and remove layers of encryption, and finally how to homomorphically compute XOR.

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

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

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

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

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

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

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

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

Error Analysis Every time we add a layer, the error increases. Hence, we need to ensure that the error does not increase too much. After l steps, the error in the public key is $\mathtt{pk}_{0\ldots l} = \sum_{i=0}^{l} \mathbf{e}_i$, where $\mathbf{e}_i$ is the error added in each step.

The error in the ciphertext is $c_{0\ldots l} = \sum_{i=0}^{l} \mathbf{x}_i \sum_{j=0}^{i} \mathbf{e}_j$, where the $\mathbf{x}_i$ is the chosen randomness in each step. Since $\mathbf{x}_i \in \{0,1\}^m$, the error in the ciphertext can be bounded by $m \cdot \max_i\{\left|\mathbf{e}_i\right|_\infty\} \cdot l^2$, which is quadratic in the number of steps.

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

Algorithm $\mathsf{hXor}(c = (c_1, c_2), c' = (c'_1, c'_2), \mathtt{pk})$

Set $c'' = (c_1 + c'_1, c_2 + c'_2)$.
Choose $\mathbf{x} \in \{0,1\}^m$ uniformly at random.
return $\mathsf{Rand}(c'', \mathtt{pk}, \mathbf{x})$

2.3.2 Efficient Topology-Hiding Computation with FHE

One thing to note is that compiling MPC from broadcast is rather expensive, especially in the fail-stop model; we need a broadcast for every round. However, we will show that an FHE scheme with additive overhead can be used to evaluate any polynomial-time function f in a topology-hiding manner. Additive overhead applies to ciphertext versus plaintext sizes and to error with respect to all homomorphic operations if necessary. We will employ an altered random walk protocol, and the total number of rounds in this protocol will amount to that of a single broadcast. We remark that FHE with additive overhead can be obtained from subexponential iO and subexponentially secure OWFs (probabilistic iO), as shown in [32].

Intuitively, we use the FHE scheme to add a layer by encrypting the ciphertext (and corresponding public key). Because the scheme is fully homomorphic, no matter how many layers there are, we are able to homomorphically evaluate any functionality through all of the layers.

To describe our method in more detail, we define a variant of a PKCR scheme, called DFH-PKE, and show how to instantiate such a scheme using FHE.

Deeply Fully-Homomorphic Public-Key Encryption

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

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

Deleting a layer is the opposite of adding a layer. The algorithm $\mathsf{DelLayer}_r$ deletes a layer from a level-$(r+1)$ ciphertext. It takes as input a vector of level-$(r+1)$

ciphertexts $[\![m]\!]_{\mathbf{pk}}$, the corresponding level-$(r+1)$ public key $\mathbf{pk}$, and the secret key $\mathtt{sk}$ that corresponds to the secret key used to add the last layer. It then outputs a vector of level-r ciphertexts and the level-r public key.

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

To describe the security of our enhanced encryption scheme, we will require that there exists an algorithm $\mathsf{Leveled\text{-}Encrypt}_r$, which takes as input a plain message and a level-r public key, and outputs a level-r ciphertext. We will then require that from the output of $\mathsf{AddLayer}_r$ ($\mathsf{DelLayer}_r$) one cannot obtain any information on the underlying layers of encryption. That is, that the output of $\mathsf{AddLayer}_r$ ($\mathsf{DelLayer}_r$) is indistinguishable from a level-$(r+1)$ (level-r) encryption of the message, i.e., the output of $\mathsf{Leveled\text{-}Encrypt}_{r+1}$ ($\mathsf{Leveled\text{-}Encrypt}_r$) under a level-$(r+1)$ (level-r) public key. We will also require that the output of HomOp_r is indistinguishable from a level-r encryption of the output of the functions applied to the messages.

We refer to Appendix 2.3.2 for a formal definition of a DFH-PKE scheme and to Appendix 2.3.2 from an instantiation from FHE.

Topology Hiding Computation from DFH-PKE.

We will use DFH-PKE to alter the $\mathsf{RandomWalkPhase}$ protocol (and by extension we can also alter the protocol $\mathsf{ProbabilisticRandomWalkPhase}_p$). Then, executing protocols $\mathsf{BC\text{-}OB}$ and $\mathsf{BC\text{-}FB}_p$ that leak one bit and a fraction of a bit respectively will be able to evaluate any poly-time function instead, while still leaking the same amount of information as a broadcast using these random walk protocols. The concept is simple. During the Aggregate Stage, parties will add a leveled encryption of their input and identifying information to a vector of ciphertexts, while adding a layer — we will not need sequential id's if each party knows where their input should go in the function. Then, at the end of the Aggregate Stage, nodes homomorphically evaluate f', which is the composition of a parsing function, to get one of each inputs in the right place, and f, to evaluate the function on the parsed inputs. The result is a leveled ciphertext of the output of f. This ciphertext is un-layered in the Decrypt Stage so that by the end, the relevant parties get the output. We refer to Appendix 2.3.2 for a more detailed description of the protocol $\mathsf{DFH\text{-}THC}$.

We will provide the formal theorem statement for the soundness of the FHE topology-hiding computation statement, and sketch the proof.

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

Sketch. This proof will look almost exactly like the proof of Theorem 1. The simulator and its use of the leakage oracle will behave in nearly the same manner as before.

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

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

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

Remarks. The definition of a DFH-PKE scheme can be relaxed to only being able to homomorphically evaluate the OR. It turns out that this relaxation is implied by any OR-homomorphic PKCR scheme and is strong enough to prove the security of the protocols $\mathsf{BC\text{-}OB}$ and $\mathsf{BC\text{-}FB}_p$. However, for simplicity, we decided to describe our protocols $\mathsf{BC\text{-}OB}$ and $\mathsf{BC\text{-}FB}_p$ from a OR-homomorphic PKCR scheme.

Deeply Fully-Homomorphic Public-Key Encryption

In this section we present the formal definition of deeply fully-homomorphic public-key encryption from Section 2.3.2. For completeness, we first give the definition of a public-key encryption scheme.

Definition 1. *A* public-key encryption *(PKE) scheme with public-key space $\mathcal{PK}$, secret-key space $\mathcal{SK}$, plaintext message space $\mathcal{M}$, and ciphertext space $\mathcal{C}$ consists of three algorithms* ($\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}$) *where:*

1. *The (probabilistic)* key-generation *algorithm $\mathsf{KeyGen} : \{0,1\}^* \to \mathcal{PK} \times \mathcal{SK}$ takes a security parameter and outputs a public key $\mathtt{pk} \in \mathcal{PK}$ and a secret key $\mathtt{sk} \in \mathcal{SK}$.*
2. *The (probabilistic)* encryption *algorithm $\mathsf{Encrypt} : \mathcal{PK} \times \mathcal{M} \to \mathcal{C}$ takes a public key $\mathtt{pk} \in \mathcal{PK}$ and a message $m \in \mathcal{M}$ and outputs a ciphertext $\mathtt{c} \in \mathcal{C}$.*
3. *The* decryption *algorithm $\mathsf{Decrypt} : \mathcal{SK} \times \mathcal{C} \to \mathcal{M}$ takes a secret key $\mathtt{sk} \in \mathcal{SK}$ and a ciphertext $\mathtt{c} \in \mathcal{C}$ and outputs a message $m \in \mathcal{M}$.*

A PKE scheme is correct *if for any key pair $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{KeyGen}$ and any message $m \in \mathcal{M}$ it holds (with probability 1 over the randomness of $\mathsf{Encrypt}$) that $m = \mathsf{Decrypt}(\mathtt{sk}, \mathsf{Encrypt}(\mathtt{pk}, m))$.*

Our protocol requires a PKE scheme $\mathcal{E}$ where (a) one can add and remove layers of encryption, while (b) one can homomorphically compute any function on encrypted bits (independent of the number of layers). This will be captured by three additional

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

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

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

For convenience, it will be easy to describe the security of our enhanced encryption scheme with the help of an algorithm $\mathsf{Leveled\text{-}Encrypt}_r$, which takes as input a vector of plain messages and a level-r public key, and outputs a vector of level-r ciphertexts[2].

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

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

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

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

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

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

$$\approx_c$$

$$\left\{ (\mathit{Leveled\text{-}Encrypt}_{r+1}(\mathbf{m}, \mathbf{pk'_2}; U^*), \mathbf{pk'_2}) : \begin{array}{c} (\mathit{pk}, \mathit{sk}) \leftarrow \mathit{KeyGen}(1^\kappa; U^*), \\ ([\![0]\!]_{\mathbf{pk'_2}}, \mathbf{pk'_2}) \leftarrow \mathit{AddLayer}_r([\![0]\!]_{\mathbf{pk_2}}, \mathbf{pk_2}, \mathit{sk}; U^*) \end{array} \right\}$$

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

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

$$\approx_c$$

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

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

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

$$\approx_c$$

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

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

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

Instantiation of DFH-PKE from FHE

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

Notation. We denote by $\mathrm{FHE}.\mathsf{Encrypt}(m, \mathtt{pk})$ the FHE encryption algorithm that takes message m and encrypts under public key $\mathtt{pk}$. In the same way, the FHE decryption algorithm is denoted by $\mathrm{FHE}.\mathsf{Decrypt}$. The FHE evaluation algorithm is defined as $\mathrm{FHE}.\mathsf{HomOp}([m_1, \ldots, m_n]_{\mathtt{pk}}, \mathtt{pk}, f) \coloneqq [f(m_1, \ldots, m_n)]_{\mathtt{pk}}$. It gets as input a vector of encrypted messages under $\mathtt{pk}$, the public key $\mathtt{pk}$ and the function to evaluate, and it returns the output of f applied to the messages.

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

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

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

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

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

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

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

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

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

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

Lemma 2. *For any r, algorithm HomOp_r is correct on leveled ciphertexts.*

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

For the base case, consider $r = 1$. Here we go into the if statement, and the algorithm returns

$$\mathrm{FHE.}\mathsf{HomOp}([\mathbf{m}]_{\mathtt{pk}}, \mathtt{pk}, f) = [f(\mathbf{m})]_{\mathtt{pk}}$$

by the correctness of the FHE scheme.

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

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

by correctness of the FHE homomorphic evaluation. □

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

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

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

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

Lemma 3. $\mathit{Leveled\text{-}Encrypt}_r(\mathbf{m}, \mathbf{pk}_r) = \llbracket \mathbf{m} \rrbracket_{\mathbf{pk}_r}$.

Proof. We will prove this by induction on r. First, when $r = 1$, we have that $\mathsf{Leveled\text{-}Encrypt}_1(\mathbf{m}, \mathbf{pk}_1) = \mathrm{FHE.Encrypt}(\mathbf{m}, \mathtt{pk}_1) = \llbracket \mathbf{m} \rrbracket_{\mathbf{pk}_1}$ from the base case.

Now, assume that for $r-1$, $\mathsf{Leveled\text{-}Encrypt}_{r-1}(\mathbf{m}, \mathbf{pk}_{r-1}) = \llbracket \mathbf{m} \rrbracket_{\mathbf{pk}_{t-1}}$. Hence, when we invoke $\mathsf{Leveled\text{-}Encrypt}_r(\mathbf{m}, \mathbf{pk}_r)$, we have that $f(\mathbf{m}, \mathbf{pk}_{r-1}) = \llbracket \mathbf{m} \rrbracket_{\mathbf{pk}_{r-1}}$ from the inductive hypothesis. Therefore, we return

$$\mathrm{FHE.HomOp}([\mathbf{m}]_{\mathtt{pk}_r}, [\mathbf{pk}_{r-1}]_{\mathtt{pk}_r}, \mathtt{pk}_r, f) = [\llbracket \mathbf{m} \rrbracket_{\mathbf{pk}_{r-1}}]_{pk_r} = \llbracket \mathbf{m} \rrbracket_{\mathbf{pk}_r}$$

as desired. □

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

Proof. **Aggregate Soundness.** The algorithm $\mathsf{AddLayer}$ returns a tuple that contains $(\llbracket \mathbf{m} \rrbracket_{\mathbf{pk}}, \mathbf{pk})$, where $\mathbf{m}$ is a vector of messages, and $\mathbf{pk}$ is a pair containing a fresh key $\mathtt{pk}$ and an encryption of a level-r key under $\mathtt{pk}$. The tuple that consists of $(\mathsf{Leveled\text{-}Encrypt}_r(\mathbf{m}, \mathbf{pk_1}; U^*), \mathbf{pk_1})$, where $\mathbf{pk_1}$ is a level-$(r+1)$ public key obtained from adding a fresh layer to a level-r public key, has the same distribution: the first part of both tuples contain fresh FHE encryptions of level-r ciphertexts.

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

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

Topology Hiding Computation from DFH-PKE

In this section, we present a detailed description of protocol DFH-THC from Section 2.3.2.

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

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

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

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

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

Round $r \in [2, T]$. Let $k = \pi_i^{(r)}(j)$. Upon receiving a vector of level-$(r-1)$ ciphertexts from P_j. Party P_i uses $\mathtt{sk}_{i\to k}^{(r)}$ to add a fresh layer with AddLayer to the vector of ciphertexts. The function AddLayer will return the vector $\mathbf{c}$ of

level-r ciphertexts with the corresponding level-r public key $\mathbf{pk}$. Then, P_i will encrypt its own input, id and unhappybit via $\mathsf{Leveled\text{-}Encrypt}$ under $\mathbf{pk}$ and appends these ciphertexts to $\mathbf{c}$. It then sends to P_k the level-r public key and all the level-r ciphertexts.

If no vector of ciphertexts was received from P_j (i.e. P_j aborted), P_i generates a fresh level-r public key $\mathbf{pk}$ and secret key $\mathbf{sk}$. It then generates a vector of level-r ciphertexts containing the bit 1 using $\mathsf{Leveled\text{-}Encrypt}$ under $\mathbf{pk}$. The size of this vector corresponds to the size of the vector containing the dummy bit, r input bits, r ids, and r unhappy bits.

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

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

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

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

Now, $\mathsf{DFH\text{-}THC}$ runs the protocol $\mathsf{DFH\text{-}RandomWalkPhase}$ n times, similarly to $\mathsf{BC\text{-}OB}$.

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

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

2.4 PKCR* Encryption

This section formally defines PKCR*—the extended Privately Key Commutative and Rerandomizable (PKCR) encryption of [6].

Let $\mathcal{PK}$, $\mathcal{SK}$ and $\mathcal{C}$ denote the public key, secret key and ciphertext spaces. In contrast to PKCR, the message space is $\{0,1\}$. Moreover, $\mathcal{C} \cap \{0,1\} = \varnothing$. As in any public-key encryption scheme, we have the algorithms $\mathsf{PKCR*.KGen} : \{0,1\}^* \to \mathcal{PK} \times \mathcal{SK}$ and $\mathsf{PKCR*.Enc} : \{0,1\} \times \mathcal{PK} \to \mathcal{C}$ for key generation and encryption, respectively (decryption can be implemented via deleting layers). Moreover, we require the following properties, where only the first two are provided (with minor differences) by PKCR.

Key-Commutative. $\mathcal{PK}$ forms a commutative group under the operation $\circledast$. In particular, given any $\mathtt{pk}_1, \mathtt{pk}_2 \in \mathcal{PK}$ and the secret key $\mathtt{sk}_1$ corresponding to $\mathtt{pk}_1$, we can efficiently compute $\mathtt{pk}_3 = \mathtt{pk}_1 \circledast \mathtt{pk}_2 \in \mathcal{PK}$ (note that $\mathtt{sk}_1$ can be replaced by $\mathtt{sk}_2$, since $\mathcal{PK}$ is commutative).

This group must interact well with ciphertexts; there exists a pair of deterministic efficiently computable algorithms $\mathsf{PKCR*.AddLayer} : \mathcal{C} \times \mathcal{SK} \to \mathcal{C}$ and $\mathsf{PKCR*.DelLayer} : \mathcal{C} \times \mathcal{SK} \to \mathcal{C} \cup \{0,1\}$ such that for every pair of public keys $\mathtt{pk}_1, \mathtt{pk}_2 \in \mathcal{PK}$ with corresponding secret keys $\mathtt{sk}_1$ and $\mathtt{sk}_2$, for every bit $b \in \{0,1\}$, and every ciphertext $c = \mathsf{PKCR*.Enc}(b, \mathtt{pk}_1)$, with overwhelming probability it holds that:

- The ciphertext $\mathsf{PKCR*.AddLayer}(c, \mathtt{sk}_2)$ is an encryption of b under the public key $\mathtt{pk}_1 \circledast \mathtt{pk}_2$.
- $\mathsf{PKCR*.DelLayer}(c, \mathtt{sk}_2)$ is an encryption of b under the public key $\mathtt{pk}_1 \circledast \mathtt{pk}_2^{-1}$.
- $\mathsf{PKCR*.DelLayer}(c, \mathtt{sk}_1) = b$.

Notice that we need the secret key to perform these operations.[3]

Rerandomizable. There exists an efficient probabilistic algorithm $\mathsf{PKCR*.Rand} : \mathcal{C} \to \mathcal{C}$, which re-randomizes a ciphertext.[4] Formally, we require that for every public key $\mathtt{pk} \in \mathcal{PK}$, every bit b, and every $c = \mathsf{PKCR*.Enc}(b, \mathtt{pk})$, the following distributions are computationally indistinguishable:

$$\{(b, c, \mathtt{pk}, \mathsf{PKCR*.Enc}(b, \mathtt{pk}))\} \approx \{(b, c, \mathtt{pk}, \mathsf{PKCR*.Rand}(c, \mathtt{pk}))\}$$

Transforming a 0-ciphertext to a 1-ciphertext. There exists an efficient algorithm $\mathsf{PKCR*.ToOne} : \mathcal{C} \to \mathcal{C}$, such that for every $\mathtt{pk} \in \mathcal{PK}$ and for every $c = \mathsf{PKCR*.Enc}(0, \mathtt{pk})$, the output of $\mathsf{PKCR*.ToOne}(c)$ is an encryption of 1 under $\mathtt{pk}$.

[3] In PKCR of [4], computing $\mathtt{pk}_1 \circledast \mathtt{pk}_2$ does not require the secret key. Moreover, PKCR requires perfect correctness.

[4] In [4] the rerandomization algorithm is given the public key as input. We also note that they require public keys to be re-randomizable, while we do not need this property.

Key anonymity. A ciphertext reveals no information about which public key was used in encryption. Formally, we require that PKCR* is key-indistinguishable (or IK-CPA secure), as defined by Bellare et al. [19].

Construction of PKCR* Based on DDH

We use a cyclic group $G = \langle g \rangle$. We keep as ciphertext a pair of group elements (c_1, c_2). The first group element contains the message. The second group element contains the secret keys of each layer of encryption. All information is contained in the exponent.

To add a layer of encryption with a secret key $\mathtt{sk}$, one simply raises the second element to $\mathtt{sk}$. Similarly, one can remove layers of encryption. When all layers of encryption are removed, both group elements are either equal $c_1 = c_2$ (the message is 0) or $c_1 = c_2^2$ (the message is 1). To transform an encryption of 0 to an encryption of 1, one simply squares the first group element.

Algorithm PKCR*

We let G be a group of order p, generated by g. These parameters are implicitly passed to all algorithms (formally, they are part of each ciphertext and an input to key generation).

PKCR*.KGen

1: Sample the secret key $\mathtt{sk}$ uniform at random from $\mathbb{Z}_p$.
2: Output $(g^{\mathtt{sk}}, \mathtt{sk})$.

PKCR*.Enc(b, y)

1: Sample r at random from $\mathbb{Z}_p$.
2: Output $c = (g^{(b+1)r}, y^r)$.

PKCR*.AddLayer$((c_1, c_2), \mathtt{sk})$

1: Output $(c_1, c_2^{\mathtt{sk}})$.

PKCR*.Rand$((c_1, c_2))$

1: Sample r at random from $\mathbb{Z}_p$.
2: Output (c_1^r, c_2^r).

PKCR*.DelLayer$((c_1, c_2), \mathtt{sk})$

1: Set $c_2' = c_2^{\mathtt{sk}^{-1}}$.
2: **if** $c_1 = c_2'$ **then** Output 0.
3: **else if** $c_1 = c_2'^2$ **then** Output 1.
4: **else** Output (c_1, c_2').

PKCR*.ToOne$((c_1, c_2))$

1: Output (c_1^2, c_2).

Security. Semantic security and KI-CPA security of our scheme follow from the respective properties of the ElGamal encryption (for the proof of KI-CPA security, see [19]). Further, the proof that it satisfies the requirements of rerandomizability and key commutativity is analogous to the proof that the DDH-based construction of [4] satisfies these properties. We refer to [4] for details.

It remains to prove the correctness of PKCR*.ToOne and PKCR*.DelLayer. The former follows trivially from inspection of the protocol.

For the latter, we need to show that the probability of PKCR*.DelLayer giving the wrong output (either from the wrong domain, or the incorrect decryption) is negligible. Observe that, by correctness of the ElGamal cryptosystem, whenever PKCR*.DelLayer should output a bit, it indeed outputs the correct value. Now, for a fixed secret key $\mathtt{sk}$, and a public key $\mathtt{pk}$, consider the probability of PKCR*.DelLayer

outputting a bit when it should output a ciphertext. This event happens only when $c_1 = c_2^{\mathtt{sk}^{-1}}$ or $c_1 = c_2^{2\mathtt{sk}^{-1}}$, which happens with probability $2/p$.

2.5 Fail-Stop Adversaries

Blurb about FS adversaries and the model.

2.5.1 Fail-Stop Model

A failstop adversary can statically select a set of parties to passively or even semi-maliciously corrupt, and during the protocol, the adversary can also adaptively crash any number of parties.

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

A *semi-malicious* corruption (see, e.g., [11]) is a stronger variant of a passive corruption. Again, we assume that the adversary selects the set of semi-malicious parties $\mathcal{Z}^s$ before the protocol execution. These parties follow the protocol instructions, but the adversary can access their internal state and can additionally choose their randomness.

In all of our results we assume a *fail-stop* adversary that can crash adaptively arbitrary parties. After being crashed, a party stops sending messages. Note that crashed parties are not necessarily corrupted. In particular, the adversary has no access to the internal state of a crashed party unless it is in the set of corrupted parties. This type of fail-stop adversary is stronger and more general than the one used in [14], where only passively corrupted parties can be crashed. In particular, in our model the adversary does not necessarily learn the neighbors of crashed parties, whereas in [14] they are revealed to it by definition.

Communication Model

We state our results in the UC framework. Following the approach in [108], to model the restricted communication network we define the $\mathcal{F}_{\text{NET}}$-hybrid model. The $\mathcal{F}_{\text{NET}}$ functionality takes as input a description of the graph network from a special "graph party" P_{setting} and then returns to each party P_i a description of its neighborhood. After that, the functionality acts as an "ideal channel" that allows parties to communicate with their neighbors according to the graph network.

Similarly to [14], we change the $\mathcal{F}_{\text{NET}}$ functionality from [108] to deal with a fail-stop adversary. However, our version differs from the one in [14] in two aspects. First, we do not strengthen the functionality $\mathcal{F}_{\text{NET}}$ in the real world to notify the neighbors of a crashed party, even though we deal with adversaries that are beyond semi-honest. Secondly, as mentioned in Section **??**, our model does not assume that only passively corrupted parties can be crashed.

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

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

Initialization Step:

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

Communication Step:

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

Observe that since $\mathcal{F}_{\text{NET}}$ gives local information about the network graph to all corrupted parties, any ideal-world adversary should also have access to this information. For this reason, [108] introduces the functionality $\mathcal{F}_{\text{INFO}}$, which contains only the Initialization Step of $\mathcal{F}_{\text{NET}}$. $\mathcal{F}_{\text{INFO}}$ is then made available to the simulator.

As shown in [108], in the fail-stop model and if the graph can potentially be disconnected, leaking some information about the network topology is inherent. We follow the approach in [14] and model the leakage by allowing the adversary access to an oracle, which, once during the protocol execution, evaluates a (possibly probabilistic) function $\mathcal{L}$. The inputs to $\mathcal{L}$ include the network graph, the set of crashed parties and arbitrary input from the adversary.

We will say that a protocol leaks one bit of information if the leakage function $\mathcal{L}$ outputs one bit. We also consider the notion of leaking a fraction p of a bit. This is modeled by having $\mathcal{L}$ output the bit only with probability p (otherwise, $\mathcal{L}$ outputs a special symbol $\perp$). Here our model differs from the one in [14], where in case of the fractional leakage, $\mathcal{L}$ always gives the output, but the simulator is restricted to query its oracle with probability p over its randomness. As noted in [14], the formulation we use is stronger.

The leakage function $\mathcal{L}$ is a parameter of the functionality $\mathcal{F}_{\text{INFO}}$ (we denote our information functionality by $\mathcal{F}_{\text{INFO}}$) $\mathcal{F}_{\text{INFO}}$ consists of two phases: an initialization phase as in $\mathcal{F}_{\text{NET}}$ and a leakage phase, where the adversary can query the leakage oracle $\mathcal{L}$.

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

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

Initialization Step:

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

Leakage Step:

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

2.5.2 Security Model

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

Technically, we model such ability in the UC framework in a way very similar to [14]. First, the ideal world adversary receives from the ideal functionality the outputs of the corrupted parties. Then, it inputs to the functionality an *abort vector* containing a list of parties who do not receive the output.

Definition 5. *We say that a protocol* Π *topology-hidingly realizes a functionality* $\mathcal{F}$ *with* $\mathcal{L}$*-leakage, in the presence of an adversary who can statically passive corrupt and adaptively crash any number of parties, if it UC-realizes* $(\mathcal{F}_{\text{INFO}} \parallel \mathcal{F})$ *in the* $\mathcal{F}_{\text{NET}}$*-hybrid model.*

2.5.3 Fail-Stop Protocols

In this section, we provide a protocol that securely realizes the broadcast functionality $\mathcal{F}_{\text{BC}}$ (with abort) in the $\mathcal{F}_{\text{NET}}$-hybrid world with at most one bit leakage. The protocol is secure in the presence of an adversary that corrupts statically passively any number of nodes and can also crash adaptively any number of nodes. It works under the DDH, the QR or the LWE assumption. We then extend this protocol to leak at most any non-negligible fraction of a bit.

If no crashes occur, both protocols do not leak any information.

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

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

As in [3], our protocols actually compute the OR of the input bits of the parties. To achieve broadcast, every party inputs the bit 0, except for the sender, who inputs the broadcast bit.

Protocol Leaking One Bit

We now present a topology-hiding broadcast protocol that is secure (with abort) against an adversary that can passively corrupt and adaptively crash parties, and leaks at most one bit of information about the network topology. If the adversary only does passive corruptions, no leakage occurs.

Our broadcast protocol BC-OB builds upon the random-walk idea from the work [3]. To prevent the adversary from learning too much information (by crashing a party), we do not use a single random walk phase. Instead, we use n consecutive random-walk phases, where in each phase only one party gets an output. We note that the same idea is used in the recent work [14].

In our protocol every party P_i holds an *unhappy-bit* u_i. Initially, every party P_i is happy, i.e., $u_i = 0$. If a neighbor of P_i crashes, then in the next phase P_i becomes unhappy and sets $u_i = 1$.

As said above, the protocol consists of n phases. In each phase, the parties execute the random-walk protocol from [3] with the following modifications.

Happiness Indicator: Instead of a single encrypted bit, parties send an encrypted tuple $[b, u]$ along the random walk. Bit u is the OR of the unhappy-bits of parties in the walk, while b is the OR of their input bits and their unhappy-bits. In other words, a party P_i on the walk homomorphically ORs $b_i \vee u_i$ to b and u_i to u. If all parties on the walk were happy at the time of adding their bits, the bit b will actually contain the proper OR of their input bits. On the other hand, if a party was unhappy, the bits u and b will actually be set to 1.

Single Output Party: In phase i all parties except P_i start their random walks with encryptions of 1 instead of their input bits. This ensures that the outputs they get from the random walk will be 1. Hence, they do not learn any information in this phase. Party P_i, on the other hand, will start exactly one random walk with its actual input bit. This ensures (in case no party crashes) that P_i actually learns the broadcast bit.

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

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

Initialization Stage:

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

2: Each party P_i generates $\mathtt{T}-1$ random permutations on d_i elements $\left\{\pi_i^{(2)}, \ldots, \pi_i^{(\mathtt{T})}\right\}$.
3: For each party P_i, if any of P_i's neighbors crashed in any phase before the current one, then P_i becomes unhappy, i.e., sets $u_i = 1$.

Aggregate Stage: Each party P_i does the following:

1: // AddLayer and HomOR applied component-wise
2: // Send first ciphertexts
3: **if** P_i is the recipient P_0 **then**
4: Party P_i sends to the first neighbor the ciphertext $[b_i \vee u_i, u_i]_{\mathtt{pk}_{i\to 1}^{(1)}}$ and the public key $\mathtt{pk}_{i\to 1}^{(1)}$.
5: Party P_i sends to any other neighbor P_j ciphertext $[1,1]_{\mathtt{pk}_{i\to j}^{(1)}}$ and the public key $\mathtt{pk}_{i\to j}^{(1)}$.
6: **else**
7: Party P_i sends to each neighbor P_j ciphertext $[1,1]_{\mathtt{pk}_{i\to j}^{(1)}}$ and the public key $\mathtt{pk}_{i\to j}^{(1)}$.
8: // Add layer while ORing own input bit
9: **for** any round r from 2 to $\mathtt{T}$ **do**
10: For each neighbor P_j of P_i, do the following:
11: **if** P_i did not receive a message from P_j **then**
12: Let $k = \pi_i^{(r)}(j)$.
13: Party P_i sends ciphertext $[1,1]_{\mathtt{pk}_{i\to k}^{(r)}}$ and public key $\mathtt{pk}_{i\to k}^{(r)}$ to neighbor P_k.
14: **else**
15: Let $k = \pi_i^{(r)}(j)$. Let $\mathtt{c}_{j\to i}^{(r-1)}$ and $\overline{\mathtt{pk}}_{j\to i}^{(r-1)}$ be the ciphertext and the public key P_i received from P_j.
16: Party P_i computes $\overline{\mathtt{pk}}_{i\to k}^{(r)} = \overline{\mathtt{pk}}_{j\to i}^{(r-1)} \circledast \mathtt{pk}_{i\to k}^{(r)}$ and $\hat{\mathtt{c}}_{i\to k}^{(r)} \leftarrow \mathsf{AddLayer}\left(\mathtt{c}_{j\to i}^{(r-1)}, \mathtt{sk}_{i\to k}^{(r)}\right)$.
17: Party P_i computes $[b_i \vee u_i, u_i]_{\overline{\mathtt{pk}}_{i\to k}^{(r)}}$ and $\mathtt{c}_{i\to k}^{(r)} = \mathsf{HomOR}\left([b_i \vee u_i, u_i]_{\overline{\mathtt{pk}}_{i\to k}^{(r)}}, \hat{\mathtt{c}}_{i\to k}^{(r)}, \overline{\mathtt{pk}}_{i\to k}^{(r)}\right)$.
18: Party P_i sends ciphertext $\mathtt{c}_{i\to k}^{(r)}$ and public key $\overline{\mathtt{pk}}_{i\to k}^{(r)}$ to neighbor P_k.

Decrypt Stage: Each party P_i does the following:

1: // DelLayer and HomOR applied component-wise
2: // Return ciphertexts
3: For each neighbor P_j of P_i:
4: **if** P_i did not receive a message from P_j at round T of the Aggregate Stage **then**
5: Party P_i sends ciphertext $\mathtt{e}_{i\to j}^{(\mathtt{T})} = [1,1]_{\overline{\mathtt{pk}}_{j\to i}^{(\mathtt{T})}}$ to P_j.
6: **else**
7: Party P_i computes and sends $\mathtt{e}_{i\to j}^{(\mathtt{T})} = \mathsf{HomOR}\left([b_i \vee u_i, u_i]_{\overline{\mathtt{pk}}_{j\to i}^{(\mathtt{T})}}, \mathtt{c}_{j\to i}^{(\mathtt{T})}, \overline{\mathtt{pk}}_{j\to i}^{(\mathtt{T})}\right)$ to P_j.
8: // Remove layers

for any round r from $\mathtt{T}$ to 2 **do**
    For each neighbor P_k of P_i:
    if P_i did not receive a message from P_k **then**
        Party P_i sends $\mathsf{e}_{i\to j}^{(r-1)} = [1,1]_{\overline{\mathtt{pk}}_{j\to i}^{(r-1)}}$ to neighbor P_j, where $k = \pi_i^{(r)}(j)$.
    else
        Denote by $\mathsf{e}_{k\to i}^{(r)}$ the ciphertext P_i received from P_k, where $k = \pi_i^{(r)}(j)$.
        Party P_i sends $\mathsf{e}_{i\to j}^{(r-1)} = \mathsf{DelLayer}\left(\mathsf{e}_{k\to i}^{(r)}, \mathtt{sk}_{i\to k}^{(r)}\right)$ to neighbor P_j.
// Only the recipient has a proper output
if P_i is the recipient P_o and happy **then**
    Party P_i computes $(b,u) = \mathsf{Decrypt}(\mathsf{e}_{1\to i}^{(1)}, \mathtt{sk}_{i\to 1}^{(1)})$.
    Party P_i **outputs** (b,u,u_i).
else
    Party P_i **outputs** $(1,0,u_i)$.

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

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

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

The protocol BC-OB leaks information about the topology of the graph during the execution of the protocol RandomWalkPhase, in which the first crash occurs. (Every execution before the first crash proceeds almost exactly as the protocol in [3] and in every execution afterwards all values are blinded by the unhappy-bit u.) We model the leaked information by a query to the leakage function $\mathcal{L}_{OB}$. The function outputs only one bit and, since the functionality $\mathcal{F}_{\text{INFO}}$ allows for only one query to the leakage function, the protocol leaks overall one bit of information.

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

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

if for any $(P_f, r) \in F$, $P_f \notin \mathcal{C}$ **then** Return 0.
else
Generate in G a random walk of length $\mathtt{T}'$ starting at P_s.
Return 1 if for any $(P_f, r) \in F$ removing party P_f after r rounds disconnects the walk and 0 otherwise.

Theorem 4. *Let κ be the security parameter. For $\mathtt{T} = 8n^3(\log(n) + \kappa)$ the protocol BC-OB($\mathtt{T}, (d_i, b_i)_{P_i \in \mathcal{P}}$)) topology-hidingly realizes $\mathcal{F}_{\text{INFO}}^{OB} || \mathcal{F}_{\text{BC}}$ (with abort) in the $\mathcal{F}_{\text{NET}}$ hybrid-world, where the leakage function $\mathcal{L}_{OB}$ is the one defined as above. If no crashes occur, then there is no abort and there is no leakage.*

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

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

Soundness. We now need to show that no environment can distinguish between the real world and the simulated world, when given access to the adversarially-corrupted parties.

Simulator. In essence, the task of $\mathcal{S}_{OB}$ is to simulate the messages sent by honest parties to passively corrupted parties. Below, we present the pseudocode of the simulator. The essential part of it is the algorithm PhaseSimulation, which is also illustrated in Figure 2-1.

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

1. $\mathcal{S}_{OB}$ corrupts passively $\mathcal{Z}^p$.
2. $\mathcal{S}_{OB}$ sends inputs for all parties in $\mathcal{Z}^p$ to $\mathcal{F}_{\text{BC}}$ and receives the output bit b^{out}.
3. For each $P_i \in \mathcal{Z}^p$, $\mathcal{S}_{OB}$ receives $\mathbf{N}_G(P_i)$ from $\mathcal{F}_{\text{INFO}}$.
4. Throughout the simulation, if $\mathcal{A}$ crashes a party P_f, so does $\mathcal{S}_{OB}$.
5. Now $\mathcal{S}_{OB}$ has to simulate the view of all parties in $\mathcal{Z}^p$.

   In every phase in which P_o should get the output, first of all the Initialization Stage is executed among the parties in $\mathcal{Z}^p$ and the $\mathtt{T}$ key pairs are generated for every $P_i \in \mathcal{Z}^p$. Moreover, for every $P_i \in \mathcal{Z}^p$ the permutations $\pi_i^{(r)}$ are generated, defining those parts of all random walks, which pass through parties in $\mathcal{Z}^p$.

The messages sent by parties in $\mathcal{Z}^p$ are generated by executing the protocol RandomWalkPhase. The messages sent by correct parties $P_i \notin \mathcal{Z}^p$ are generated by executing PhaseSimulation(P_o, P_i), described below.

6. $\mathcal{S}_{OB}$ sends to $\mathcal{F}_{\text{BC}}$ the abort vector (in particular, the vector contains all parties P_o who should receive their outputs in phases following the first crash and, depending on the output of $\mathcal{L}_{OB}$, the party who should receive its output in the phase with first crash).

Algorithm PhaseSimulation(P_o, P_i)

If $P_o \in \mathcal{Z}^p$, let w denote the random walk generated in the Initialization Stage (at the beginning of the simulation of this phase), which starts at P_o and carries the output bit. Let ℓ denote the number of parties in $\mathcal{Z}^p$ on w before the first correct party. If $P_o \notin \mathcal{Z}^p$, w and ℓ are not defined.
For every $P_j \in \mathcal{Z}^p \cap \mathbf{N}_G(P_i)$, let $\mathtt{pk}_{j\to i}^{(r)}$ denote the public key generated in the Initialization Stage by P_j for P_i and for round r .

Initialization Stage

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

Aggregate Stage

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

Decrypt Stage

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

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

We first argue why this simulator simulates correctly the real world. Consider a corrupted party P_c and its honest neighbor P_h. The messages sent from P_h to P_c during the Aggregate Stage are ciphertexts, to which P_h added a layer, and corre-

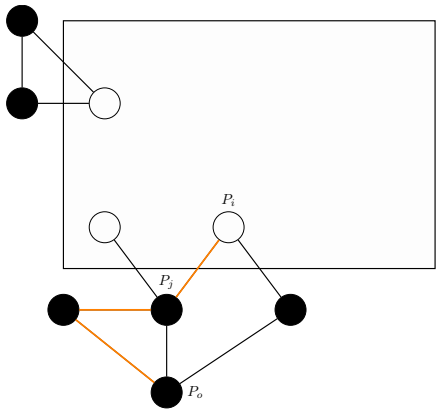

Figure 2-1: An example of the algorithm executed by the simulator $\mathcal{S}_{OB}$. The filled circles are the corrupted parties. The red line represents the random walk generated by $\mathcal{S}_{OB}$ in Step 5, in this case of length $\ell = 3$. $\mathcal{S}_{OB}$ simulates the Decrypt Stage by sending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

sponding public keys. Since P_h is honest, the adversary does not know the secret keys corresponding to the sent public keys. Hence, $\mathcal{S}_{OB}$ can simply replace them with encryptions of a pair $(1, 1)$ under a freshly generated public key. The group structure of keys in PKCR guarantees that a fresh key has the same distribution as the composed key (after executing AddLayer). Semantic security implies that the encrypted message can be replaced by $(1, 1)$.

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

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

For this one meaningful message, we consider three cases. If any party crashed in a phase preceding the current one, $\mathcal{S}_{OB}$ sends an encryption of $(1, 1)$ (as in the real world the walk is made dummy by an unhappy party). If no crashes occurred up to this point (round r in given phase), $\mathcal{S}_{OB}$ encrypts the output received from $\mathcal{F}_{\text{BC}}$. If a crash happened in the current phase, $\mathcal{S}_{OB}$ queries the leakage oracle $\mathcal{L}_{OB}$, which executes the protocol and tells whether the output or $(1, 1)$ should be sent, essentially telling the simulator if the relevant random walk was disrupted by the crash or not.

Hybrids and security proof. We present a description of the hybrids and security

proof.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- A phase where the adversary did not crash any party in this or any previous phase. In this case, the simulator $\mathcal{S}_3$ sends an encryption of (b_W, u_W), where $b_W = \bigvee_{P_j \in W} b_j$ is the OR of all input bits in the walk and $u_W = 0$, since no crash occurred. $\mathcal{S}_4$ sends an encryption of $(b^{out}, 0)$, where $b^{out} = \bigvee_{P_i \in \mathcal{P}} b_i$. Since the graph is connected, $b^{out} = b_W$ with overwhelming probability, as proven in Lemma 1. Also, the encryption in Hybrid 4 is done with a fresh public key which is indistinguishable with the encryption done in Hybrid 3 by OR'ing many times in the graph, as shown in Claim 2.1 in [3].

- A phase where the adversary crashed a party in a previous phase or any round different than $2\mathtt{T} - \ell$ of the first phase where the adversary crashes a party. In Hybrid 4 the parties send an encryption of $(1, 1)$. This is also the case in Hybrid 3, because even if a crashed party disconnected the graph, each connected component contains a neighbor of a crashed party. Moreover, in Hybrid 4, the messages are encrypted with a fresh public key, and in Hybrid 3, the encryptions are obtained by the homomorphic OR operation. Both encryptions are indistinguishable, as shown in in Claim 2.1 in [3].

■

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

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

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

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

■

This concludes the proof of soundness.

□ □

Protocol Leaking a Fraction of a Bit

In this section we present a broadcast protocol $\mathsf{BC\text{-}FB}_p$, which leaks only a fraction of a bit of information about the network topology. If no crashes occur during the execution of $\mathsf{BC\text{-}FB}_p$, then no information about the topology is leaked and all parties receive the correct output with overwhelming probability.

Formally, fractional leakage is modeled by an oracle, which is very similar to the one presented in Section 2.5.3 for the protocol $\mathsf{BC\text{-}OB}$, however, it answers its query only with probability p. That is, with probability $1 - p$ the response is $\perp$ (in this case no information leaks) and with probability p it is either 0 or 1, according to the leaked information.

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

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

The probability p is a parameter of our protocol. In other words, we can tolerate arbitrarily low leakage. However, the communication and round complexity of our

protocol are proportional to $1/p$. As a consequence, $1/p$ must be polynomial and the leakage probability p cannot be negligible.

The Protocol BC-FB$_p$. The idea is to leverage the fact that the adversary can gain information from crashing fail-stop corrupted parties only in one execution of RandomWalkPhase. (This is because a phase with a crash causes an abort, that is, all subsequent phases produce no outputs and reveal no information.) Imagine that RandomWalkPhase can randomly fail with probability $1-p$. A failure means that an execution does not give any outputs nor any information (similarly to the behavior in case of an abort). Moreover, assume that a failure becomes apparent (both to the honest parties and to the adversary) only at the last moment, when the output is computed. Of course, we now have to repeat the phase a number of times, in order to preserve correctness. However, the adversary has only one chance to choose the execution of RandomWalkPhase, in which it crashes the first party, at the same time losing the ability to learn anything from the subsequent phases. Moreover, it must choose it before it knows whether the execution failed. Since the chance of failure is $1-p$, the adversary learns any information only with probability p.

More formally, the protocol BC-FB$_p$, similarly to protocol BC-OB, consists of n random-walk phases. However, each of the n phases is modified so that with probability at least $1-p$ it fails, and then repeated ρ times (for sufficiently large ρ). We now describe the modification, which makes a random-walk phase fail. The change concerns only the Aggregate Stage. The idea is that instead of an encrypted tuple $[b,u]$, $\lfloor 1/p \rfloor + 1$ encrypted values $[b^1,\ldots,b^{\lfloor 1/p \rfloor},u]$ are sent along the walk. The bit u is the OR of the unhappy-bits of the parties on the walk. Every bit b^k is equivalent to the bit b in the protocol RandomWalkPhase. That is, b^k is the OR of the input bits and the unhappy-bits of the parties. However, all but one of the $\lfloor 1/p \rfloor$ bits b^k are initially set to 1 by the party who starts the walk. This means that only one of the bits b^k is not dummy, but without knowing the secret key it is impossible to tell which one. Throughout the Aggregate Stage, the parties process every ciphertext corresponding to a bit b^k the same way they processed the encryption of b in RandomWalkPhase. The encryption of the unhappy-bit u is also processed the same way. However, before sending the ciphertexts to the next party on the walk, the encryptions of the bits b^k are randomly shuffled. This way we make sure that the adversary does not know which of the ciphertexts contain dummy values (the adversary cannot tell which values are dummy as soon as they have been permuted by at least one non-passively corrupted party). At the end of the Aggregate Stage (after T rounds), the last party chooses uniformly at random one of the $\lfloor 1/p \rfloor$ ciphertexts and uses it, together with the encryption of the unhappy-bit, to execute the Decrypt Stage as in RandomWalkPhase.

After ρ repetitions of the phase, a party P_o does the following in order to get the output. If in any repetition the unhappy-bit was set to 1, P_o cannot compute the output (that is, an abort was detected). Otherwise, P_o outputs the AND of all bits output by individual repetitions.

Overall, the global parameters passed to BC-FB$_p$ are the length T of the random walk and the number ρ of repetitions of a phase. As in BC-OB, each party P_i has

input (d_i, b_i, u_i), where d_i is its number of neighbors, u_i is its the unhappy-bit, and b_i is the input bit.

A formal description of the modified protocol $\mathsf{ProbabilisticRandomWalkPhase}_p$ for the random-walk phase can be found in Appendix 2.5.4. Note that this protocol should be repeated ρ times in the actual protocol.

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

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

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

Soundness. We show that no environment can distinguish between the real world and the simulated world, when given access to the adversarially-corrupted nodes. The simulator $\mathcal{S}_{FB}$ for $\mathsf{BC\text{-}FB}_p$ is a modification of $\mathcal{S}_{OB}$. Here we only sketch the changes and argue why $\mathcal{S}_{FB}$ simulates the real world. A formal proof of the soundness of $\mathsf{BC\text{-}FB}_p$ can be found in Appendix 2.5.5.

In each of the ρ repetitions of a phase, $\mathcal{S}_{FB}$ executes a protocol very similar to the one for $\mathcal{S}_{OB}$. In the Aggregate Stage, $\mathcal{S}_{FB}$ proceeds almost identically to $\mathcal{S}_{OB}$ (except that it sends encryptions of vectors $(1, \ldots, 1)$ instead of only two values). In the Decrypt Stage the only difference between $\mathcal{S}_{FB}$ and $\mathcal{S}_{OB}$ is in computing the output for the party P_o (as already discussed in the proof of Theorem 4, $\mathcal{S}_{FB}$ does this only when P_o is corrupted and the walk carrying the output enters an area of corrupted parties). In the case when there were no crashes before or during given repetition of a phase, $\mathcal{S}_{OB}$ would simply send the encrypted output. On the other hand, $\mathcal{S}_{FB}$ samples a value from the Bernoulli distribution with parameter p and sends the encrypted output only with probability p, while with probability $1 - p$ it sends the encryption of $(1, 0)$. Otherwise, the simulation is the same as for $\mathcal{S}_{OB}$.

It can be easily seen that $\mathcal{S}_{FB}$ simulates the real world in the Aggregate Stage and in the Decrypt Stage in every message other than the one encrypting the output. But even this message comes from the same distribution as the corresponding message

sent in the real world. This is because in the real world, if the walk was not broken by a crash, this message contains the output with probability p. The simulator encrypts the output also with probability p in the two possible cases: when there was no crash ($\mathcal{S}_{FB}$ samples from the Bernoulli distribution) and when there was a crash but the walk was not broken ($\mathcal{L}_{FB}$ is defined in this way). □

This section contains supplementary material for Section **??**.

2.5.4 Protocol Leaking a Fraction of a Bit

In this section we give a formal description of the random-walk phase $\mathsf{ProbabilisticRandomWalkPhase}_p$ for the broadcast protocol $\mathsf{BC\text{-}FB}_p$ from Section 2.5.3. The boxes indicate the parts where it differs from the random-walk phase protocol $\mathsf{RandomWalkPhase}$ for the broadcast protocol leaking one bit (cf. Section 2.5.3).

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

Initialization Stage:

Each party P_i generates $\mathtt{T} \cdot d_i$ keypairs $(\mathtt{pk}_{i\to j}^{(r)}, \mathtt{sk}_{i\to j}^{(r)}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ where $r \in \{1, \ldots, \mathtt{T}\}$ and $j \in \{1, \ldots, d_i\}$.
Each party P_i generates $\mathtt{T} - 1$ random permutations on d_i elements $\left\{\pi_i^{(2)}, \ldots, \pi_i^{(\mathtt{T})}\right\}$.
For each party P_i, if any of P_i's neighbors crashed in any phase before the current one, then P_i becomes unhappy, i.e., sets $u_i = 1$.

Aggregate Stage: Each party P_i does the following:

// Send first ciphertexts
if P_i is the recipient P_0 **then**
Party P_i sends to the first neighbor the public key $\mathtt{pk}_{i\to 1}^{(1)}$ and the ciphertext $\boxed{[b_i \vee u_i, 1, \ldots, 1, u_i]_{\mathtt{pk}_{i\to 1}^{(1)}}}$ ($\lfloor 1/p \rfloor$ ciphertexts contain $b_i \vee u_i$).
Party P_i sends to any other neighbor P_j ciphertext $\boxed{[1, \ldots, 1, 1]_{\mathtt{pk}_{i\to j}^{(1)}}}$ and the public key $\mathtt{pk}_{i\to j}^{(1)}$.
else
Party P_i sends to each neighbor P_j ciphertext $\boxed{[1, \ldots, 1, 1]_{\mathtt{pk}_{i\to j}^{(1)}}}$ and the public key $\mathtt{pk}_{i\to j}^{(1)}$.
// Add layer while ORing own input bit
for any round r from 2 to $\mathtt{T}$ **do**
For each neighbor P_j of P_i, do the following:
if P_i did not receive a message from P_j **then**
Let $k = \pi_i^{(r)}(j)$.
Party P_i sends ciphertext $\boxed{[1, \ldots, 1, 1]_{\mathtt{pk}_{i\to k}^{(r)}}}$ and public key $\mathtt{pk}_{i\to k}^{(r)}$ to neighbor P_k.
else

Let $k = \pi_i^{(r)}(j)$. Let $\mathsf{c}_{j\to i}^{(r-1)}$ and $\overline{\mathsf{pk}}_{j\to i}^{(r-1)}$ be the ciphertext and the public key P_i received from P_j.

Party P_i computes $\overline{\mathsf{pk}}_{i\to k}^{(r)} = \overline{\mathsf{pk}}_{j\to i}^{(r-1)} \circledast \mathsf{pk}_{i\to k}^{(r)}$ and $\hat{\mathsf{c}}_{i\to k}^{(r)} \leftarrow \mathsf{AddLayer}\left(\mathsf{c}_{j\to i}^{(r-1)}, \mathsf{pk}_{i\to k}^{(r)}\right)$.

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

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

Decrypt Stage: Each party P_i does the following:

// Return ciphertexts For each neighbor P_j of P_i:

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

Party P_i sends ciphertext $\mathsf{e}_{i\to j}^{(\mathtt{T})} = [1,1]_{\overline{\mathsf{pk}}_{j\to i}^{(\mathtt{T})}}$ to P_j.

else

Party P_i chooses uniformly at random one of the first $\lfloor 1/p \rfloor$ ciphertexts in $\mathsf{c}_{j\to i}^{(\mathtt{T})}$. Let $\bar{\mathsf{c}}_{j\to i}^{(\mathtt{T})}$ denote the tuple containing the chosen ciphertext and the last element of $\mathsf{c}_{j\to i}^{(\mathtt{T})}$ (the encryption of the unhappy bit). Party P_i computes and sends $\mathsf{e}_{i\to j}^{(\mathtt{T})} = \mathsf{HomOR}\left([b_i \vee u_i, u_i]_{\overline{\mathsf{pk}}_{j\to i}^{(\mathtt{T})}}, \bar{\mathsf{c}}_{j\to i}^{(\mathtt{T})}\right)$ to neighbor P_j.

// Remove layers

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

For each neighbor P_k of P_i:

if P_i did not receive a message from P_k **then**

Party P_i sends $\mathsf{e}_{i\to j}^{(r-1)} = [1,1]_{\overline{\mathsf{pk}}_{j\to i}^{(r-1)}}$ to neighbor P_j, where $k = \pi_i^{(r)}(j)$.

else

Denote by $\mathsf{e}_{k\to i}^{(r)}$ the ciphertext P_i received from P_k, where $k = \pi_i^{(r)}(j)$.

Party P_i sends $\mathsf{e}_{i\to j}^{(r-1)} = \mathsf{DelLayer}\left(\mathsf{e}_{k\to i}^{(r)}, \mathsf{sk}_{i\to k}^{(r)}\right)$ to neighbor P_j.

// Only the recipient has a proper output

if P_i is the recipient P_o and happy **then**

Party P_i computes $(b,u) = \mathsf{Decrypt}(\mathsf{e}_{1\to i}^{(1)}, \mathsf{sk}_{i\to 1}^{(1)})$.

Party P_i **outputs** (b, u, u_i).

else

Party P_i **outputs** $(1, 0, u_i)$.

2.5.5 Soundness of the Protocol Leaking a Fraction of a Bit

In this Section we show the soundness of $\mathsf{BC\text{-}FB}_p$ from Section 2.5.3. That is, we present the simulator, the hybrids and the security proof for Theorem 1.

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

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

Proof. (soundness part)

Simulator. The simulator $\mathcal{S}_{FB}$ proceeds almost identically to the simulator $\mathcal{S}_{OB}$ given in the proof of Theorem 4. We only change the algorithm PhaseSimulation to ProbabilisticPhaseSimulation and execute it ρ times instead of only once.

Algorithm ProbabilisticPhaseSimulation(P_o, P_i)

If $P_o \in \mathcal{Z}^p$, let w denote the random walk generated in the Initialization Stage (at the beginning of the simulation of this phase), which starts at P_o and carries the output bit. Let ℓ denote the number of parties in $\mathcal{Z}^p$ on w before the first correct party. If $P_o \notin \mathcal{Z}^p$, w and ℓ are not defined.
For every $P_j \in \mathcal{Z}^p \cap \mathbf{N}_G(P_i)$, let $\mathtt{pk}_{j\to i}^{(r)}$ denote the public key generated in the Initialization Stage by P_j for P_i and for round r .

Initialization Stage

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

Aggregate Stage

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

Decrypt Stage

if $P_o \notin \mathcal{Z}^p$ or $\mathcal{A}$ crashed any party in any phase before the current one or in any repetition of the current phase **then**

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

else

In every round r and for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{FB}$ sends $[1,1]_{\mathtt{pk}_{j\to i}^{(r)}}$ to P_j unless the following three conditions hold: (a) P_i is the first party not in $\mathcal{Z}^p$ on w, (b) P_j is the last party in $\mathcal{Z}^p$ on w, and (c) $r = 2\mathtt{T} - \ell$.

If the three conditions hold (in particular $r = 2\mathtt{T} - \ell$), $\mathcal{S}_{FB}$ does the following. If $\mathcal{A}$ did not crash any party in a previous round,

$\mathcal{S}_{FB}$ samples a value x from the Bernoulli distribution with parameter p'. If $x = 1$ (with probability p'), $\mathcal{S}_{FB}$ sends to P_j the ciphertext $[b_{out}, 0]_{pk_{j\to i}^{(r)}}$ and otherwise it sends $[1,0]_{pk_{j\to i}^{(r)}}$.

Otherwise, let F denote the set of pairs $(P_f, s-\ell+1)$ such that $\mathcal{A}$ crashed P_f in round s. $\mathcal{S}_{FB}$ queries $\mathcal{F}_{\text{INFO}}^{FB_p}$ for the leakage on input $(F, P_i, \mathtt{T}-\ell)$. If the returned value is 1, it sends $[1,1]_{\mathtt{pk}_{j\to i}^{(r)}}$ to P_j. Otherwise it sends $[b^{out}, 0]_{\mathtt{pk}_{j\to i}^{(r)}}$ to party P_j.

Hybrids and security proof. We consider similar steps as the hybrids from Paragraph 2.5.3.

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

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

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

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

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

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

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

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

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

- A subphase l of a phase k where the adversary did not crash any party in this phase, any previous subphase, or any previous phase. In this case, $\mathcal{S}_3$ sends with probability p an encryption of (b_W, u_W), where $b_W = \bigvee_{u \in W} b_u$ is the OR

of all input bits in the walk and $u_W = 0$ (since no crash occurs), and with probability $1-p$ an encryption of $(1,0)$. On the other hand, $\mathcal{S}_4$ samples r from a Bernoulli distribution with parameter p, and if $r = 1$, it sends an encryption of $(b_{out}, 0)$, where $b_{out} = \bigvee_{i\in[n]} b_i$, and if $r = 0$ it sends an encryption of $(1,0)$. Since the graph is connected, $b_{out} = b_W$ with overwhelming probability, as proven in Lemma 1. Also, the encryption in Hybrid 4 is done with a fresh public key which is indistinguishable with the encryption done in Hybrid 3 by OR'ing many times in the graph, as shown in Claim 2.1. in [3].

- A subphase l of a phase k where the adversary crashed a party in a previous subphase or a previous phase.

  In Hybrid 3 the parties send encryptions of $(1,1)$. This is also the case in Hybrid 4, because even if a crashed party disconnected the graph, each connected component contains a neighbor of a crashed party. Moreover, in Hybrid 4, the messages are encrypted with a fresh public key, and in Hybrid 3, the encryptions are obtained by the homomorphic OR operation. Both encryptions are indistinguishable, as shown in Claim 2.1. in [3].

■

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

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

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

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

This concludes the proof of soundness. □ □

2.6 Security Against Semi-malicious Adversaries

In this section, we present a generic idea for turning any topology-hiding protocol secure against an adversary, who can statically passively corrupt and adaptively crash parties into one secure against a semi-malicious adversary (who can also adaptively crash parties). The core idea of our transformation is to first generate "good" random tapes for all parties and then execute the given protocol with parties using the pre-generated random tapes instead of generating randomness on the fly. Hence, the transformed protocol proceeds in two phases: Randomness Generation and Deterministic Execution.

Randomness Generation. The goal of the first phase is to generate for each party P_i a uniform random value r_i, which can then be used as randomness tape of P_i in the phase of Deterministic Execution. To generate those random values the parties use the following flooding scheme over n rounds.

Protocol GenerateRandomness

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

Lemma 5. *Let G' be the network graph without parties which crashed during the execution of the protocol* GenerateRandomness*. Any party P_i whose connected component in G' contains at least one honest party will output a uniform value r_i. The output of any honest party is not known to the adversary. The protocol* GenerateRandomness *does not leak any information about the network-graph (even if crashes occur).*

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

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

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

Finally, observe that the message pattern seen by a party is determined by its neighborhood. Moreover, the messages received by corrupted parties from honest

parties are uniform random values. This implies, that the view of the adversary in this protocol can be easily simulated given the neighborhood of corrupted parties. Thus, the protocol does not leak any information about the network topology. □ □

Transformation to Semi-Malicious Security. In the second phase of Deterministic Execution, the parties execute the protocol secure only against passive and fail-stop corruptions, but instead of generating fresh randomness during the protocol execution, the parties use the random tape generated in the first phase. If a party witnessed a crashed in the first phase, it pretends the this crash happens in the first round of the actual protocol and proceeds accordingly.

Protocol EnhanceProtocol(Π)

1. The parties execute GenerateRandomness to generate random tapes.
2. The parties execute the actual protocol:
   - Parties use the generated randomness tapes instead of generating randomness on the fly.
   - If a party witnessed a crashed in the GenerateRandomness, it pretends that it witnessed this crash in the first round of the protocol Π (and proceeds in the protocol accordingly).

Theorem 2. *Let $\mathcal{F}$ be an MPC functionality and let Π be a protocol that topology-hidingly realizes $\mathcal{F}$ in the presence of static passive corruptions and adaptive crashes. Then, the protocol* EnhanceProtocol(Π) *topology-hidingly realizes $\mathcal{F}$ in the presence of static semi-malicious corruption and adaptive crashes. The leakage stays the same.*

Proof. (sketch) The randomness generation protocol GenerateRandomness used in the first phase is secure against a semi-malicious fail-stopping adversary. Lemma 5 implies that the random tape of any semi-malicious party that can interact with honest parties is truly uniform random. Moreover, the adversary has no information on the random tapes of honest parties. This implies that the capability of the adversary in the execution of the actual protocol in the second phase (which for fixed random tapes is deterministic) is the same as for an semi-honest fail-stopping adversary. This implies that the leakage of EnhanceProtocol(Π) is the same as for Π as the randomness generation protocol does not leak information (even if crashes occur). □ □

Remark 1. *To improve overall communication complexity of the protocol the values generated in the first phase could be used as local seeds for a PRG which is then used to generate the actual random tapes.*

As a corollary of Theorem 1 and 2, we obtain that any MPC functionality can be realized in a topology hiding manner against an adversary that does any number of static semi-malicious corruptions and adaptive crashes, leaking at most an arbitrary small fraction p of information about the topology.

Corollary 2. *cor:FinalIf DDH, QR or LWE is hard, then for any MPC functionality $\mathcal{F}$ there is a topology-hiding protocol realizing $\mathcal{F}$ leaking at most an arbitrarily small fraction p of a bit, which is secure against an adversary that does any number of static semi-malicious corruptions and adaptive crashes. The round and communication complexity is polynomial in κ and $1/p$.*

2.6.1 Compiling Fail-Stop Protocols with Leakage

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

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

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

Topology-hiding computation without aggregated leakage. In this section we show how to transform any protocol secure in the presence of passive and fail-stop corruptions into one that additionally hides the network topology and leaks at most an arbitrary fraction p of a bit *in total*. To achieve this, we first consider a variant of parallel composition of the broadcast protocol $\mathsf{BC\text{-}FB}_p$. That is, we show how to modify $\mathsf{BC\text{-}FB}_p$ to construct an all-to-all multibit broadcast protocol (where each party simultaneously broadcasts a k bit message), which leaks only a fraction p of a bit and has the same round complexity as $\mathsf{BC\text{-}FB}_p$. Then, we consider sequential composition of all-to-all broadcasts and note that the technique we use is in fact more general and can be applied to a wider class of protocols which leak information. With the above statements, we conclude that any protocol can be compiled into one that leaks only a fraction p of a bit in total. As a corollary, any functionality $\mathcal{F}$ can be implemented by a topology hiding protocol leaking any fraction p of a bit.

All-to-all Multibit Broadcast

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

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

Now we show how to achieve an all-to-all broadcast. Assume each party P_i wants to broadcast some k-bit message, $(b_1, \ldots, b_k)$. We consider a vector of length nk, where each of the n parties is assigned to k slots for k bits of its message. Each of the vectors $\mathbf{b_i}$ in the vector $[\mathbf{b_1}, \ldots, \mathbf{b}_\ell, u]$ described above will be of this form. P_i will use the slots from $n(i-1)$ to ni to communicate its message. This means that P_i will have as input vector $\mathbf{b_i} = (0, \ldots, 0, b_1, \ldots, b_k, 0, \ldots, 0)$. Then, in the Aggregate Stage, the parties will input their input message into their corresponding slots (by homomorphically OR'ing the received vector with its input message). At the end of the protocol, each party will receive the output containing the broadcast message of each party P_j in the slots $n(j-1)$ to nj.

More formally, we have the following Lemma, proven in Appendix 2.7.1.

Lemma 6. *Protocol $\mathsf{BC\text{-}FB}_p$ can be edited to an all-to-all multi-bit broadcast $\mathsf{MultibitBC}_p$, which is secure against an adversary, who statically passively corrupts and adaptively crashes any number of parties and leaks at most a fraction p of a bit. The round complexity of $\mathsf{MultibitBC}_p$ is the same as for $\mathsf{BC\text{-}FB}_p$.*

Sequential Execution Without Aggregated Leakage

We show how to construct a protocol, which implements any number of sequential executions of the protocol $\mathsf{MultibitBC}_p$, while preserving the leakage of a fraction p of a bit in total. The construction makes non-black-box use of the unhappy bits used in $\mathsf{MultibitBC}_p$. The idea is simply to preserve the state of the unhappy bits between sequential executions. That is, once some party sees a crash, it will cause all subsequent executions to abort. For a proof of the following lemma, we refer to Appendix 2.7.2.

Lemma 7. *There exists a protocol, which implements any number k of executions of $\mathsf{MultibitBC}_p$, is secure against an adversary, who statically passively corrupts and adaptively crashes any number of parties and leaks at most a fraction p of a bit in total. The complexity of the constructed protocol is k times the complexity of $\mathsf{MultibitBC}_p$.*

Remark 2. *We note that the above technique to sequentially execute protocols which leak p bits and are secure* with abort *can be applied to a more general class of protocols (in particular, not only to our topology-hiding broadcast). The idea is that if a protocol satisfies the property that any abort before it begins implies that the protocol does not leak any information, then it can be executed sequentially leaking at most p bits.*

Topology-Hiding Computation

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

To do this, it is enough to do a standard transformation using public key infrastructure. Let Π_{MPC} be a protocol that runs in M rounds. First, the parties use one all-to-all multi-bit topology hiding broadcast protocol to send each public key to every other party. Then, each round of Π_{MPC} is simulated: the parties run n all-to-all multi-bit topology hiding broadcasts simultaneously to send the messages sent in that round encrypted under the corresponding public keys. After the broadcasts, each party can use their secret key to decrypt their corresponding messages. For a proof of the following theorem, we refer to Appedix 2.7.3.

Theorem 5. *Assume* PKCR *exists. Then, we can compile any MPC protocol* Π_{MPC} *that runs in* M *rounds into a topology-hiding protocol with leakage function* $\mathcal{L}_{FB_p}$*, that runs in* $MR+1$ *rounds, where* R *is the round complexity of* $\mathsf{BC\text{-}FB}_p$*.* [5]

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

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

Proof. Because every poly-time computable functionality $\mathcal{F}$ has an MPC protocol [33], we get that Theorem 5 implies we can get topology-hiding computation. The round and communication complexity is implied by Theorem 5 and the complexity of $\mathsf{MultibitBC}_p$. □ □

2.7 Appendix for Getting Topology-Hiding Computation

This section contains supplementary material for Section 2.6.1.

2.7.1 All-to-all Multibit Broadcast

In this section we prove Lemma 6 from Section 2.6.1 which shows how to modify protocol $\mathsf{BC\text{-}FB}_p$ to achieve an all-to-all multibit broadcast.

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

Lemma 6. *Protocol $\mathsf{BC\text{-}FB}_p$ can be edited to an all-to-all multi-bit broadcast $\mathsf{MultibitBC}_p$, which is secure against an adversary, who statically passively corrupts and adaptively crashes any number of parties and leaks at most a fraction p of a bit. The round complexity of $\mathsf{MultibitBC}_p$ is the same as for $\mathsf{BC\text{-}FB}_p$.*

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

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

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

- Every party P_i that wants to send the k bit message $(b_1, \ldots, b_k)$ prepares its input vector
$$\mathbf{b_i} = (0, \ldots, 0, b_1, \ldots, b_k, 0, \ldots, 0)$$
by placing the bits $b_1, \ldots, b_k$ in positions from $n(i-1)$ to ni.

- At the beginning of the Aggregate Stage, the recipient P_o with the input vector $\mathbf{b_o}$ sends the ciphertext $[\mathbf{b_o} \vee u_o, \mathbf{1}, \ldots, \mathbf{1}, u_o]_{\mathtt{pk}^{(1)}_{i\to 1}}$ to its first neighbor. All other ciphertexts to all other neighbors j are just $[\mathbf{1}, \ldots, \mathbf{1}, 1]_{\mathtt{pk}^{(1)}_{i\to j}}$[6].

  Every other party P_i starts the protocol with sending the ciphertext tuple $[\mathbf{1}, \ldots, \mathbf{1}, 1]_{\mathtt{pk}^{(1)}_{i\to j}}$ to every neighbor j.

- Upon receiving a ciphertext at round r from a neighbor j, $[\mathbf{b}_1, \ldots, \mathbf{b}_\ell, u]_{pk^{(t)}_{j\to i}}$, party P_i takes its input vector $\mathbf{b_i}$ and homomorphically OR's the vector $(\mathbf{b_i} \vee u_i, \ldots, \mathbf{b_i} \vee u_i, u_i)$ containing ℓ copies of the vector $\mathbf{b_i} \vee u_i$ to the ciphertext. The result is sent along the walk.

The rest of the protocol $\mathsf{MultibitBC}_p$ proceeds analogously to $\mathsf{BC\text{-}FB}_p$.

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

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

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

2.7.2 Sequential Execution Without Aggregated Leakage

In this section we prove Lemma 7 from Section 2.6.1 which states that there is a protocol that executes $\mathsf{MultibitBC}_p$ sequentially any number of times leaking at most a fraction p of a bit.

Lemma 7. *There exists a protocol, which implements any number k of executions of $\mathsf{MultibitBC}_p$, is secure against an adversary, who statically passively corrupts and adaptively crashes any number of parties and leaks at most a fraction p of a bit in total. The complexity of the constructed protocol is k times the complexity of $\mathsf{MultibitBC}_p$.*

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

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

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

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

2.7.3 Topology-Hiding Computation

In this section we prove Theorem 5 from Section 2.6.1 which compiles any MPC protocol secure against an adversary who can statically passively corrupt and adaptively crash any number of parties into a protocol that is in addition topology-hiding and leaks at most a fraction p of a bit of information about the topology.

Theorem 5. *Assume PKCR exists. Then, we can compile any MPC protocol Π_{MPC} that runs in M rounds into a topology-hiding protocol with leakage function $\mathcal{L}_{FB_p}$, that runs in $MR+1$ rounds, where R is the round complexity of $\mathsf{BC\text{-}FB}_p$.* [7]

Proof. Recall the generic transformation for taking UC-secure topology-hiding broadcast and compiling it into UC-secure topology-hiding MPC using a public key infrastructure. Every MPC protocol with M rounds, Π_{MPC}, has at each round each party

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

sending possibly different messages to every other party. This is a total of $O(n^2)$ messages sent at each round, but we can simulate this with n separate multi-bit broadcasts.

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

- Setup phase. The parties use one multi-bit topology-hiding broadcast to give their public key to every other party.

- Each round of Π_{MPC}. For each party P_i that needs to send a message of k bits to party P_j, P_i encrypts that message under P_j's public key. Then, each party P_i broadcasts the $n-1$ messages it would send in that round of Π_{MPC}, one for each $j \neq i$, encrypted under the appropriate public keys. That is, P_i is the source for one multi-bit broadcast. All these multi-bit broadcasts are simultaneously executed via an all-to-all multi-bit broadcast, where each party broadcast a message of size $(n-1)k$ times.
  After the broadcasts, each node can use their secret key to decrypt the messages that were for them and continue with the protocol.

- At the end of the protocol, each party now has the output it would have received from running Π_{MPC}, and can compute its respective output.

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

Security of this construction (and, in particular, the fact that it only leaks a fraction p of a bit) follows directly from Lemma 6 and Lemma 7. □

□

2.8 Delays and Topology Hiding

2.8.1 Contributions

This motivates the goal of this work, which is to construct THC protocols for more realistic settings, where messages are not guaranteed to be delivered within a fixed time bound.

Model. A natural starting point would be to consider the strongest possible adversary, i.e. one who fully controls message delivery (this is the standard setting considered by asynchronous MPC, e.g. [26, 31]). First, note that this standard model is not well suited for our setting, since in order to decide when messages are delivered, the adversary must know the network, which we attempt to hide. The next logical step is to consider a model where the adversary can only interfere with delays between parties he controls, but unfortunately, even this grants the adversary too much power. In fact, we prove in Appendix 2.10 that it is impossible to get a topology-hiding broadcast in this model.

This forces us to define a slightly weaker model. We call it the Probabilistic Unknown Delay Model and we formally define it in Section 2.9. In this model the messages are delayed independently of the adversary, but different connections have different, unbounded probabilistic delays. This means that we throw off the assumption that makes the synchronous protocols impractical. Still, parties have access to synchronized clocks.

Protocols. We remark that it is not easy to modify synchronous THC protocols (even those tolerating fail-stop adversaries) to remain secure in the Probabilistic Unknown Delay Model. For example, consider the standard technique of letting each party attach to each message a round number r, and then wait until it receives all round-r messages before proceeding to the next round. This seems to inherently leak the topology, as the time at which a party receives a message for round r reveals information about the neighborhood of the sender (e.g., that it contains an edge with very long delays).

This forces us to develop new techniques, which result in three new protocols, secure in the Probabilistic Unknown Delay Model against any number of passive corruptions. We require a setup, but this setup is independent of the network topology (it only depends on the number of parties), and it can be used to run multiple instances of the protocols, with different communication graphs.

Our first two protocols (Section 2.11.1) implement topology-hiding broadcast (any functionality can then be realized using standard techniques, by executing a sequence of broadcasts). The protocols are based on standard assumptions, but can only be used in limited classes of graphs (the same ones as in [6]): cycles and trees, respectively.[8] Furthermore, observe that the running time of a protocol could itself leak information about the topology. Indeed, this issue seems very difficult to overcome, since, intuitively, making the running time fully independent of the graph delays conflicts with our goal to design protocols that run as fast as the *actual* network. We deal with this by making the running time of our protocols depend only on the sum of all the delays in the network.

Then, in Section 2.14, we introduce a protocol that implements any functionality, works on arbitrary connected graphs, and its running time corresponds to (one sample of) the sum of all delays. On the other hand, we assume stateless secure hardware.

[8] Our second protocol works for any graphs, as long as we agree to reveal a spanning tree: the parties know which of their edges are on the tree and execute the protocol, ignoring other edges. See also [6].

Intuitively, a hardware box is a stateless program with an embedded secret key (the same for all parties). This assumption was introduced in [14] in order to deal with fail-stop adversaries in THC. Similar assumptions have also been considered before, for example, stateless tamper-proof tokens [38, 67, 45][9], or honestly-generated secure hardware [76, 44].

While secure hardware is a very strong assumption, the paradigm of constructing protocols with the help of a hardware oracle and then replacing the hardware oracle by more standard assumptions is common in the literature (see for example the secure hardware box assumption for the case of synchronous topology-hiding computation (with known upper bounds on the delays) for fail-stop adversaries [14], which was later relaxed to standard assumptions [94], or the Signature Card assumption for proofs-carrying-data schemes [44]). We hope that the techniques presented in this paper can be useful to construct protocols in more standard models.

2.8.2 Related Work

Topology-hiding computation was introduced by Moran et al. in [108]. The authors propose a broadcast protocol tolerating any number of passive corruptions. The construction uses a series of nested multi-party computations, in which each node is emulated by its neighbors. This broadcast protocol can then be used to achieve topology-hiding MPC using standard techniques to transform broadcast channels into secure point-to-point channels. In [74], the authors provide a more efficient construction based on the DDH assumption. However, both results are only feasible for graphs with logarithmic diameter. Topology-hiding communication for certain classes of graphs with large diameter was described in [6]. This result was finally extended to arbitrary (connected) graphs in [4]. These results were extended to the fail-stop setting in [14] based on stateless secure hardware, and [94] based on standard assumptions. All of the results mentioned above are in the cryptographic setting. Moreover, all results are stated in the synchronous communication model with known upper bounds on the delays.

In the information-theoretic setting, the main result is negative [73]: any topology-hiding MPC protocol inherently leaks information about the network graph. This work also shows that if the routing table is leaked, one can construct an MPC protocol which leaks no additional information.

2.8.3 Probabilistic Unknown Delay Model

2.9 The Probabilistic Unknown Delay Model

At a high level, we assume loosely synchronized clocks, which allow the parties to proceed in rounds. However, we do not assume that the messages are always de-

[9]The difference here is that a token typically needs to be passed around during the protocol and the parties can embed their own programs in it, whereas a secure hardware box is used only by one party and is initialized with the correct program.

livered within one round. Rather, we model channels that have delays drawn from some distributions each time a message is sent along (a different distribution for each channel). These delays are a property of the network. As already mentioned, this allows to achieve a significant speedup, comparable to that of asynchronous protocols and impossible in the fully synchronous model.

2.9.1 Impossibility of Stronger Models

Common models for asynchronous communication [26, 31] consider a worst-case scenario and give the adversary the power to schedule the messages. By scheduling the messages, the adversary automatically learns which parties are communicating. As a consequence, it is unavoidable that the adversary learns the topology of the communication graph, which we want to hide.

A natural definition, then, would be to give the adversary control over scheduling on channels from his corrupted parties. However, any reasonable model in which the adversary has the ability to delay messages for an unbounded amount of time allows him to learn something about the topology of the graph. In essence, a very long delay from a party behaves almost like an abort, and an adversary can exploit this much like a fail-stop adversary in the impossibility result of [107]. We formally prove this in a very weak adversarial model in Appendix 2.10.

Since delays cannot depend on the adversary without leaking topology, delays are an inherent property of the given network, much like in real life. As stated before, we give each edge a delay distribution, and the delays of messages traveling along that edge are sampled from this distribution. This allows us to model real-life networks where the adversary cannot tamper with the network connections. For example, on the Internet, delays between two directly connected nodes depend on their distance and the reliability of their connection.

2.9.2 Adversary

We consider an adversary, who statically and passively corrupts any set $\mathcal{Z} \subseteq \mathcal{P} = \{P_1, \ldots, P_n\}$ of parties, with $|\mathcal{Z}| < n$. Static corruptions mean that the set $\mathcal{Z}$ is chosen before the protocol execution. Passively corrupted parties follow the protocol instructions, but the adversary can access their internal states during the execution.

The setting with passive corruptions and secure hardware boxes is somewhat subtle. In particular, the adversary is allowed to input to the box of a corrupted party any messages of his choice, even based on secret states of other corrupted parties; he can even replay messages from honest parties with different corrupted inputs. This will be why we need authenticated encryption, for example. Importantly, in the passive model, the messages actually sent by a corrupted party are produced using the box with valid inputs.

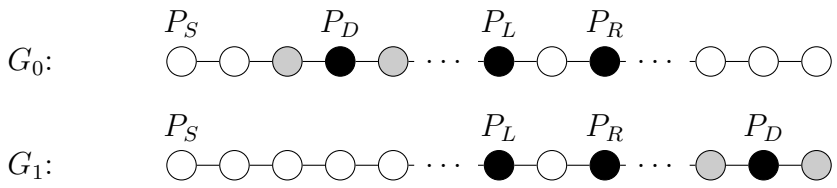

Figure 2-2: Graphs used to prove the impossibility of THC with adversarial delays. P_S is the sender. The corrupted parties (black dots) are: P_L and P_R (they delay messages), and the detective P_D. The adversary determines whether P_D (and its two neighbors) are on the left or on the right.

Impossibility of other models

2.10 Adversarially-Controlled Delays Leak Topology

Much like how adversarially-controlled aborts were shown to leak topological information in [107], we can show that adversarially-controlled delays also leak topological information. First, note that if we have bounded delays, we can always use a synchronous protocol, starting the next round after waiting the maximum delay. So, in order for this model to be interesting, we must assume the adversary has unbounded delays. In order to be as general as possible, we prove this with the weakest model we can while still giving the adversary some control over its delays: the adversary can only add delay to messages leaving corrupt nodes.

Our proof will follow the structure of [107], using a similar game-based definition and even using the same adversarially-chosen graphs (see figure 2-2). Our game is straightforward. The adversary gives the challenger two graphs and a set of corrupt nodes so that the corrupt neighborhoods are identical when there is no adversarially added delay. The challenger then chooses one of those graphs at random, runs the protocol, and gives the views of all corrupt nodes to the adversary. The adversary wins if she can tell which graph was used. In [107], the adversary would choose a round to failstop one of its corrupt parties. In our model, the adversary will instead choose a time (clock-tick) to add what we call a long-delay (which is just a very long delay on sending that and all subsequent messages). The adversary will be able to detect the delay based on when the protocol ends: if the delay was early in the protocol, the protocol takes longer to finish for all parties, and if it was late, the protocol will still finish quickly for most parties.

This impossibility result translates to an impossibility in the simulation-based setting since a secure protocol for the simulation-based setting would imply a secure protocol for the game-based setting.

2.10.1 Adversarially-Controlled Delay Indistinguishability-based Security Definition

Before proving the impossibility result, we first formally define our model. This model is as weak as possible while still assuming delays are somewhat controlled by the adversary. We will assume a minimum delay along edges: it takes at least one clock-tick for a message to get from one party to another.

Delay Algorithms In order to give the adversary as little power as possible, we define a public (and arbitrary) randomized algorithm that outputs the delays for a graph for protocol Π. Both the adversary and challenger have access to this algorithm and can sample from it.

Definition 6. *A* indistinguishability-delay algorithm (IDA) *for a protocol* Π, $\mathsf{DelayAlgorithm}_\Pi$, *is a probabilistic polynomial-time algorithm that takes as input an arbitrary graph outputs unbounded polynomial delays for every time* τ *and every edge in the graph. Explicitely, for any graph* $G = (V, E)$, $\mathsf{DelayAlgorithm}(G)$ *outputs* $\mathcal{T}$ *such that for every edge* $(i, j) \in E_b$ *and time* τ, $\mathcal{T}((i, j), \tau) = d_{(i,j),\tau}$ *is a delay that is at least one.*

The Indistinguishability Game This indistinguishability definition is a game between an adversary $\mathcal{A}$ and challenger $\mathcal{C}$ adapted from [107]. Let $\mathsf{DelayAlgorithm}$ be an IDA as defined above.

- Setup: Let $\mathcal{G}$ be a class of graphs and Π a topology-hiding broadcast protocol that works on any of the networks described by $\mathcal{G}$ according to our adversarial delay model, and let $\mathsf{DelayAlgorithm}$ be a public, fixed IDA algorithm. Without loss of generality, let P_1 have input $x \in \{0, 1\}$, the broadcast bit.

- $\mathcal{A}$ chooses two graphs $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ from $\mathcal{G}$ and then a subset $\mathcal{Z}$ of the parties to corrupt. $\mathcal{Z}$ must look locally the same in both G_0 and G_1. Formally, $\mathcal{Z} \subset V_0 \cap V_1$ and $\mathbf{N}_{G_0}(\mathcal{Z}) = \mathbf{N}_{G_1}(\mathcal{Z})$. If this doesn't hold, $\mathcal{C}$ wins automatically.
  $\mathcal{A}$ then generates $\mathcal{T}_\mathcal{Z}$, a function defining delays for every edge at every time-step controlled by the adversary. That is, $\mathcal{T}_\mathcal{Z}((i, j), \tau) = d_{(i,j),\tau}$, and if $P_i \in \mathcal{Z}$, then every message sent from P_i to P_j at time τ is delayed by an extra $d_{(i,j),\tau}$.
  $\mathcal{A}$ sends $G_0, G_1, \mathcal{Z}$, and $\mathcal{T}_\mathcal{Z}$ to $\mathcal{C}$.

- $\mathcal{C}$ chooses a random $b \in \{0, 1\}$ and executes Π in G_b with delays according to $\mathsf{DelayAlgorithm}(G_b) = \mathcal{T}$ for all messages sent from honest parties. For messages sent from corrupt parties, delay is determined by the time and parties as follows: for time τ a message sent from party $P_i \in \mathcal{Z}$ to P_j has delay $\mathcal{T}((i, j), \tau) + \mathcal{T}_\mathcal{Z}((i, j), \tau)$ in reaching P_j. $\mathcal{A}$ receives the view of all parties in $\mathcal{Z}$ during the execution.

- $\mathcal{A}$ then outputs $b' \in \{0, 1\}$ and wins if $b' = b$ and loses otherwise.

Notice that in this model, the adversary statically and passively corrupts any set of parties, and statically determines what delays to add to the protocol.

Definition 7. *A protocol* Π *is* indistinguishable under chosen delay attack (IND-CDA) *over a class of graphs* $\mathcal{G}$ *if for any PPT adversary* $\mathcal{A}$*, there exists an IDA* DelayAlgorithm *such that*

$$\Pr[\mathcal{A} \textit{ wins}] \leq \frac{1}{2} + \text{negl}(n).$$

2.10.2 Proof that Adversarially-Controlled Delays Leak Topology

First, we will define what we mean when we say a protocol is 'weakly' realized in the adversarial delay model. Intuitively, it is just that the protocol outputs the correct bit to all parties if there is no adversarial delay.

Definition 8. *A protocol* Π weakly realizes the broadcast functionality *if* Π *is such that when all parties execute honestly with delays determined by any IDA, all parties get the broadcast bit within polynomial time (with all but negligible probability).*

Theorem 6. *There does not exist an IND-CDA secure protocol* Π *that weakly realizes the broadcast functionality of any class of graphs* $\mathcal{G}$ *that contains line graphs.*

Throughout the proof and associated claim, we refer to a specific pair of graphs that the adversary has chosen to distinguish between, winning the IND-CDA game. Both graphs will be a line of n vertices: $G = (V, E)$ where $E = \{(P_i, P_{i+1})\}_{i=1,\ldots,n-1}$. We will let Π be a protocol executed on G that weakly realizes broadcast when P_1 is the broadcaster, see Figure 2-2.

Our adversary in this model will either add no delay, or will add a very long polynomial delay to every message sent after some time τ.

Notice that $\mathcal{A}$ is given access to DelayAlgorithm at the start of the protocol. One can sample from DelayAlgorithm using G_0, G_1, and $\mathcal{Z}$ to get an upper bound T on the time it takes Π to terminate with all but negligible probability. Since Π weakly realizes broadcast, T is polynomial. So, $\mathcal{A}$ has access to this upper bound T.

Long-delays. Let a long-delay be a delay that lasts for T clock-ticks. Consider an adversary that will only add long-delays to a protocol, and once an adversary has long-delayed a message, he must continue to long-delay messages along that edge until the end of the protocol. That is, once the adverary decides to delay along some edge, all subsequent messages along that edge cannot arrive for at least T clock-ticks.

Claim 7. *Consider any party* P_v *whose neighbors do not add any extra delay as described by the long-delay paragraph above. As in [107], let* $H_{v,b}$ *be the event that* P_v *outputs the broadcast bit by time* T *(*P_v *may still be running the protocol by time* T *or terminate by guessing a bit by* T*). Let* E_τ *be the event that the first long-delay is at time* τ*. Then either* Π *is not IND-CDA secure, or there exists a bit* b *such that*

$$|\Pr[H_{v,b}|E_{T-1}] - \Pr[H_{v,b}|E_0]| \geq \frac{1}{2} - \text{negl}(n).$$

Proof. If some P_i long-delays at time 0, then the first message it sends is at time T, and so the graph is disconnected until time T. This makes it impossible for parties separated from P_1 to learn about the output bit by time T. So, by that time, these parties must either guess an output bit (and be right with probability at most 1/2) or output nothing and keep running the protocol (which is still not $H_{v,b}$). If Π is IND-CDA secure, then all honest parties must have the same probability of outputting the output bit by time T, and so there exists a b such that $\Pr[H_{v,b}|E_0] \leq \frac{1}{2} - \text{negl}(n)$ for all honest parties P_v.

However, if P_i long-delays at time $T-1$, then the only parties possibly affected by P_i are P_{i-1} and P_{i+1}; all other parties will get the output by time T and the information that P_i delayed cannot reach them (recall we assumed a minimum delay of at least one clock-tick in the DelayAlgorithm). So, $\Pr[H_{v,b}|E_0] = \Pr[H_{v,b}|\text{no extra delays}] = 1 - \text{negl}(n)$ for all honest parties without a delaying neighbor by the definition of weakly realizing broadcast.

The claim follows: $|\Pr[H_{v,b}|E_{T-1}] - \Pr[H_{v,b}|E_0]| \geq |\frac{1}{2} - \text{negl}(n) - 1| \geq \frac{1}{2} - \text{negl}(n)$. □ □

Theorem 6. This just follows from the previous claim. A simple hybrid argument shows that there exists a pair $(\tau^*, b) \in \{0, \ldots, T-1\} \times \{0, 1\}$ such that

$$|\Pr[H_{v,b}|E_{\tau^*}] - \Pr[H_{v,b}|E_{\tau^*+1}]| \geq \frac{1}{2T} - \text{negl}(n)$$

for all P_v who do not have a neighbor delaying. Since T is polynomial, this is a non-negligible value. Without loss of generality, assume $\Pr[H_{v,b}|E_{\tau^*}] > \Pr[H_{v,b}|E_{\tau^*+1}]$. Leveraging this difference, we will construct an adversary $\mathcal{A}$ that can win the IND-CDA game with non-negligible probability.

$\mathcal{A}$ chooses two graphs G_0 and G_1. $G = G_0$ and G_1 is G except parties 3, 4, and 5 are exchanged with parties $n-2$, $n-1$, and n respectively. $\mathcal{A}$ corrupts the source part $P_S := P_1$, a left party $P_L := P_{n/2-1}$, a right party $P_R := P_{n/2+1}$, and the detective party $P_D := P_4$. See figure 2-2 for how this looks. The goal of $\mathcal{A}$ will be to determine if P_D is to the left or right side of the network (close to the broadcaster or far).

$\mathcal{A}$ computes the upper bound T using DelayAlgorithm and randomly guesses (τ^*, b) that satisfy the inequality above. At time τ, $\mathcal{A}$ initiates a long-delay at party P_L, and at time $\tau+1$, $\mathcal{A}$ initiates a long-delay at party P_R. So, $\mathcal{A}$ gives the challenger $\mathcal{T}_{\mathcal{Z}}$ where $\mathcal{T}_{\mathcal{Z}}((i,j),t) = 0$ for $t < \tau^*$, and for $t \geq \tau^*$: $\mathcal{T}_{\mathcal{Z}}((L, n/2), t) = \mathcal{T}_{\mathcal{Z}}((L, n/2-2), t)T$ and $\mathcal{T}_{\mathcal{Z}}((R, n/2), t+1) = \mathcal{T}_{\mathcal{Z}}((R, n/2+2), t+1) = T$.

Notice that news of P_L's delay at time τ^* cannot reach P_R or any other party on the right side of the graph by time T. Also note that the time $\mathcal{A}$ gets output for each of its corrupt parties is noted in the trasncript.

If $\mathcal{C}$ chooses G_0, then P_D is on the left side of the graph and has probability $\Pr[H_{D,b}|E_{\tau^*}]$ of having the output bit by time T because its view is consistent with P_L delaying at time τ^*. If $\mathcal{C}$ chooses G_1, then P_D is on the right side of the graph, and has a view consistent with the first long delay happening at time τ^*+1 and therefore has $\Pr[H_{D,b}|E_{\tau^*}]$ of having the output bit by time T. Because there is a noticeable difference in these probabilities, $\mathcal{A}$ can distinguish between these two cases with $\frac{1}{2}$ plus some non-negligible probability. □ □

Consequences of this lower bound. We note that this is just one model where we prove it is impossible for the adversary to control delays. However, we restrict the adversary a great deal, to the point of saying that regardless of what the natural network delays are, the adversary can learn something about the topology of the graph. The lower bound proved in this model seems to rule out any possible model (simulation or game-based) where the adversary has power over delays.

2.10.3 Protocols for Cycles and Trees

2.11 Probabilistic Unknown Delay Model Protocols

2.11.1 Protocols for Restricted Classes of Graphs

This section considers protocols that realize topology-hiding broadcast in the Probabilistic Unknown Delay Model under standard assumptions (in particular, we give an instantiation based on DDH), but in the limited setting where graphs are trees or cycles. We stress that we can deal with any graphs if a spanning tree is revealed. In the following, we first recall the known technique to achieve fully-synchronous THC using random walks and so-called PKCR encryption [4]. Then, we extend PKCR by certain additional properties, which allows us to construct a broadcast protocol for cycles in the Probabilistic Unknown Delay Model. Finally, we extend this protocol to trees.

Protocol for Cycles

We assume an enhanced PKCR scheme, denoted $\mathsf{PKCR*}$. The main differences from PKCR are as follows. First, the message space in $\mathsf{PKCR*}$ is now the set $\{0,1\}$, and it is disjoint from the ciphertext space. This allows to distinguish between a layered ciphertext and a plaintext. Moreover, we no longer require explicit homomorphism, but instead use the algorithm $\mathsf{PKCR*.ToOne}(c)$ that transforms an encryption of 0 into an encryption of 1 without knowing the public key[10]. We formally define $\mathsf{PKCR*}$ and give an instantiation based on the DDH assumption in Appendix 2.4.

Rounds.

Although we are striving for a protocol that behaves in a somewhat asynchronous way, we still have a notion of rounds defined by a certain number of clock ticks. Even though each party is activated in every clock tick, each party receives, processes and sends a message only every R clock ticks — this keeps parties in sync despite delays, without clogging the network. Even if no message is received, a message is sent[11].This

[10]Its functionality does not matter and is left undefined on encryptions of 1.

[11]If the parties do not send at every round, the topology would leak. Intuitively, when a party P_i sends the initial message to its right neighbor P_j, the right neighbor of P_j learns how big the delay from P_i to P_j was. We can extend this to larger neighborhood, eventually revealing information about relative positions of corrupted parties.

means that at time τ, we are on round $\mathsf{r}_\tau = \lfloor \tau / R \rfloor$; the τ parameter will be dropped if obvious from context. Moreover, observe that the message complexity increases as R decreases. For reference, R can be thought of as relatively large, say 1,000 or more; this is also so that parties are able to completely process messages every round.

A protocol with constant delays. To better explain our ideas, we first describe our protocol in the setting with constant delays, and then modify it to deal with any delay distributions.

The high-level idea is to execute directly the decrypt phase of the random-walk protocol, where the walk is simply the cycle traversal, and the combined public key corresponding to the ciphertext resulting from the aggregate phase is given as the setup (note that this is independent of the order of parties on the graph). More concretely, we assume that each party P_i holds a secret key $\mathtt{sk}_i$ and the combined public key $\mathtt{pk} = \mathtt{pk}_1 \circledast \ldots \circledast \mathtt{pk}_n$. Assume for the moment that each party knows who the next clockwise party is in the cycle. At the beginning, a party P_i, every round (i.e., every R clock ticks), starts a new cycle traversal by sending to the next party a fresh encryption of its input $\mathsf{PKCR}^*.\mathsf{Enc}(b_i, \mathtt{pk})$. Once P_i starts receiving ciphertexts from its neighbor (note that since the delays are fixed, there is at most one ciphertext arriving in a given round), it instead continues the cycle traversals. That is, every time it receives a ciphertext c from the previous neighbor, it deletes the layer of encryption using its secret key: $\mathsf{PKCR}^*.\mathsf{DelLayer}(c, \mathtt{sk}_i)$. It then rerandomizes the result and sends it to the next party. The sender additionally transforms the ciphertext it receives to a 1-ciphertext in case its bit is 1. After traversing the whole cycle, all layers of encryption are removed and the parties can recognize a plaintext bit. This happens at the same time for every party.

In order to remove the assumption that each party knows who the next clockwise party is, we simply traverse the cycle in both directions.

A protocol accounting for variable delays. The above approach breaks down with arbitrary delays, where many messages can arrive at the same round. We deal with this by additionally ensuring that every message is received in a predictable timely manner: we will be repeating message sends. As stated in Section 2.9, the delays could be variable, but we make the assumption that if messages are sent at least R clock-ticks from each other, then the delay for each message is independent. We also assume that the median value of the delay along each edge is polynomial, denoted as $\mathrm{Med}[D_e]$. Now, since the protocol will handle messages in rounds, the actual values we need to consider are all in rounds: $\lceil \mathrm{Med}[D_e]/R \rceil$.

Now, if over κ rounds, P_1 sends a message c each round, the probability that none of the copies arrives after $\kappa + \lceil \mathrm{Med}[D_e]/R \rceil$ rounds is negligible in terms of κ, the security parameter (see Lemma 8 for the proof). Because we are guaranteed to have the message by that time (and we believe with reasonable network delays, median delay is small), we wait until time $(\kappa + \lceil \mathrm{Med}[D_e]/R \rceil) \cdot R$ has passed from when the original message was sent before processing it.[12]

[12]Note that delays between rounds are independent, but not within the round. This means we need to send copies of the message over multiple rounds for this strategy to work.

For the purposes of this sketch, we will just consider sending messages one way around the protocol. We will also focus on P_1 (with neighbors P_n and P_2) since all parties will behave in an identical manner. First, the setup phase gives every party the combined public key $\mathtt{pk} = \mathtt{pk}_1 \circledast \ldots \circledast \mathtt{pk}_n$. At each step, processing a message will involve using the PKCR.DelLayer functionality for their key.

In the first round, P_1 sends its bit (0 if not the source node, b_s if the source node) encrypted under $\mathtt{pk}$ to P_2, let's call this message $c_1^{(1)}$. P_1 will wait $\mathsf{w} = \kappa + \lceil \mathrm{Med}[D_e]/R \rceil$ rounds to receive P_n's first message during this time. Now, because P_1 needs to make sure $c_1^{(1)}$ makes it to P_2, for the next κ rounds, P_1 continues to send $c_1^{(1)}$. However, because P_1 also needs to hide w (and thus cannot reveal when it starts sending its processed message from P_n), P_1 starts sending a new ciphertext encrypting the same message, $c_2^{(1)}$ (again κ times over κ rounds), until it has waited w rounds — so, P_1 is sending $c_1^{(1)}$ and $c_2^{(1)}$ in the second round, $c_1^{(1)}$, $c_2^{(1)}$ and $c_3^{(1)}$ the third round and so forth until it sends $c_1^{(1)}, \ldots c_\kappa^{(1)}$ in round κ. Then it stops sending $c_1^{(1)}$ and starts sending $c_{\kappa+1}^{(1)}$. P_1 will only ever send κ messages at once per round. Once it has waited w rounds, P_1 is guaranteed to have received the message from P_n and can process and forward that message, again sending it κ times over κ rounds. In the next round, P_1 will then be guaranteed to receive the next message from P_n, and so on.

Denote the median-round-sum as $\mathrm{MedRSum[D]} = \sum_{i=1}^{n} \left\lceil \mathrm{Med}[D_{(i,(i+1 \mod n)+1)}]/R \right\rceil$. Because each party waits like this, the protocol has a guaranteed time to end, the same for all parties:

$$R \cdot \sum_{i=1}^{n} \mathsf{w}_i = R\left(n\kappa + \mathrm{MedRSum[D]}\right).$$

This is the only information 'leaked' from the protocol: all parties learn the sum of ceiling'd medians, MedRSum[D]. Additionally, parties all know the (real, not a round-delay) distribution of delays for messages to reach them, and thus can compute $\lceil \mathrm{Med}[D_e]/R \rceil$ for their adjacent edges.

Formally, the protocol CycleProt is described as follows.

Protocol CycleProt

// The common input of all parties is the round length R. Additionally, the sender P_s has the input bit b_s.

Setup: For $i \in \{1, \ldots, n\}$, let $(\mathtt{pk}_i, \mathtt{sk}_i) = \mathsf{PKCR^*.KGen}(1^\kappa)$. Let $\mathtt{pk} = \mathtt{pk}_1 \circledast \ldots \circledast \mathtt{pk}_n$. The setup outputs to each party P_i its secret key $\mathtt{sk}_i$ and the product public key $\mathtt{pk}$.

Initialization for each P_i:

- Send (GetInfo) to the functionality $\mathcal{F}_{\text{NET}}$ and assign randomly the labels P^0, P^1 to the two neighbors.
- Let $\mathsf{Rec}^0, \mathsf{Rec}^1$ be lists of received messages from P^0 and P^1 respectively, both initialized to $\perp$. Let Send^0 and Send^1 be sets initialized to $\varnothing$; these are the sets of messages that are ready to be sent.

- For each $\ell \in \{0,1\}$, $D_{(i,\ell)}$ is the delay distribution on the edge between P_i and P^ℓ, obtained from $\mathcal{F}_{\text{INFO}}$.
- Let $\mathsf{w}^\ell = \kappa + \left\lceil \text{Med}[D_{(i,\ell)}]/R \right\rceil$ be the time P_i waits before sending a message from P^ℓ to $P^{1-\ell}$

Execution for each P_i:

1: Send (ReadClock) to the functionality $\mathcal{F}_{\text{CLOCK}}$ and let τ be the output. If $\tau \bmod R \neq 0$, send (Ready) to the functionality $\mathcal{F}_{\text{CLOCK}}$. Otherwise, let $\mathsf{r} = \tau/R$ be the current round number and do the following:

2: *Receive messages:* Send (FetchMessages, i) to the functionality $\mathcal{F}_{\text{NET}}$. For each message (r_c, c) received from a neighbor P^ℓ, set $\mathsf{Rec}^\ell[\mathsf{r}_c + \mathsf{w}^\ell] = c$.

3: *Process if no messages received:* For each neighbor P^ℓ such that $\mathsf{Rec}^\ell[\mathsf{r}] = \perp$, start a new cycle traversal in the direction of $P^{1-\ell}$:

- If P_i is sender (i.e. $i = s$) then add $(\kappa, \mathsf{r}, \mathsf{PKCR*.Enc}(b_s, \mathtt{pk}))$ to $\mathsf{Send}^{1-\ell}$.
- Otherwise, add $(\kappa, \mathsf{r}, \mathsf{PKCR*.Enc}(0, \mathtt{pk}))$ to $\mathsf{Send}^{1-\ell}$.

4: *Process received messages:* For each P^ℓ such that $\mathsf{Rec}^\ell[\mathsf{r}] \neq \perp$ (we have received a message from P^ℓ), set $\mathsf{d} = \mathsf{PKCR*.DelLayer}(R^\ell[\mathsf{r}], \mathtt{sk}_i)$, and do the following:

- If $\mathsf{d} \in \{0,1\}$, output d and halt (we have decrypted the source bit).
- Otherwise, if $i = s$ and $b_s = 1$, then set $\mathsf{d} = \mathsf{PKCR*.ToOne}(\mathsf{d})$. Then, in either case, add $(\kappa, \mathsf{r}, \mathsf{PKCR*.Rand}(\mathsf{d}))$ to $\mathsf{Send}^{1-\ell}$.

5: *Send message:* For each $\ell \in \{0,1\}$, let $\mathsf{Sending}^\ell = \{(k, \mathsf{r}_c, c) \in \mathsf{Send}^\ell : k > 0\}$. For each $(k, \mathsf{r}_c, c) \in \mathsf{Sending}^\ell$, send (r_c, c) to P^ℓ.

6: *Update* Send *set:* For each $(k, \mathsf{r}_c, c) \in \mathsf{Sending}^\ell$, remove (k, r_c, c) from Send^ℓ and insert $(k-1, \mathsf{r}_c, c)$ to Send^ℓ.

7: Send (Ready) to the functionality $\mathcal{F}_{\text{CLOCK}}$.

In Appendix 2.12 we prove the following theorem. ($\mathcal{F}_{\text{BC}}$ denotes the broadcast functionality.)

Theorem 7. *The protocol* CycleProt *UC-realizes* $(\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{INFO}}^{\mathcal{L}_{\mathsf{median}}}, \mathcal{F}_{\text{BC}})$ *in the* $(\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{NET}})$*-hybrid model with an adversary who statically passively corrupts any number of parties, where the leakage function is defined as* $\mathcal{L}_{\mathsf{median}}(R, \mathcal{D}) = \text{MedRSum}[\text{D}]$.[13]

2.11.2 Protocol for Trees

We show how to modify the cycle protocol presented in the previous section to securely realize the broadcast functionality $\mathcal{F}_{\text{BC}}$ in any tree. As observed in [6], given a tree, nodes can locally compute their local views of a cycle-traversal of the tree. However, to apply the cycle protocol to this cycle-traversal, we would need as setup a combined public key that has each secret key $\mathtt{sk}_i$ as many times as P_i appears in

[13]Note that the round length R is a parameter of the protocol, so we allow the adversary to provide it.

the cycle-traversal. To handle that, each party simply removes its secret key from the ciphertexts received from the first neighbor, and we can assume the same setup as in the cycle protocol.

In Appendix 2.13 we give a formal description of the protocol TreeProt. The proof of the following theorem is a straightforward extension of the proof of Theorem 7.

Theorem 8. *The protocol* TreeProt *UC-realizes* $(\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{INFO}}^{\mathcal{L}_{\mathsf{median}}}, \mathcal{F}_{\text{BC}})$ *in the* $(\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{NET}})$*-hybrid model with an adversary who statically passively corrupts any number of parties, where the leakage function is defined as* $\mathcal{L}_{\mathsf{median}}(R, \mathcal{D}) = \text{MedRSum}[\text{D}]$.

2.12 Proof of Theorem 7

Simulator. We simulate the outputs of $\mathcal{F}_{\text{NET}}$ on inputs $(\text{FETCHMESSAGES}, i)$ from the corrupted parties (note that everything else can be simulated trivially). The messages sent by the corrupted parties can be easily generated by executing the protocol. Hence, the challenge is to generate the messages sent by honest parties to their corrupted neighbors.

We first deal with the problem of outputting the messages at correct times. That is, the simulator generates all messages upfront. The messages are then stored in `buffer`, and the simulator outputs them by executing the algorithm of $\mathcal{F}_{\text{NET}}$.

What remains is to show how to compute the messages. This will be done per a corrupted arc of the cycle. Observe that a sequence of corrupted parties can fast-forward the protocol and learn the output before the protocol terminates. Concretely, consider an honest party neighboring the corrupted arc. Right before the end of the protocol, it sends messages, that can be read by its direct corrupted neighbor. Before that, it sends messages, that can be read by its colluding two-neighborhood. This continues until time t, before which the messages carry the output for a party outside of the corrupted arc. The messages sent before time t are computed as encryptions of 0 under a fresh public key, since the corrupted arc cannot decrypt these messages. The messages sent after t are encryptions of the output bit.

Finally, we need a way to compute the time t, after which the messages sent by an honest party carry output for a party in the corrupted arc. As noted in Section 2.11.1, the protocol has a *deterministic* end time of $T = R\,(n\kappa + \text{MedRSum}[\text{D}])$. Consider a single corrupted arc $P_1, \ldots, P_k$ (all corrupted arcs can be handled independently since there is at least one honest party between them). A message sent from P_k of that arc to an honest node can be read by the corrupted arc when it reaches P_1 of the arc. Since the corrupted arc knows the waiting time for its parties $(\mathsf{w}_1, \ldots, \mathsf{w}_k)$, the simulator also knows these values, and so the time at which the message is revealed to the arc is T minus the time it would have taken for that message to traverse from P_1 to P_k: $T - \sum_{i=1}^{k} \mathsf{w}_i$. This is how we compute t.

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

1. $\mathcal{S}_{cycle}$ corrupts the parties in the set $\mathcal{Z}$.
2. $\mathcal{S}_{cycle}$ sends inputs for all parties in $\mathcal{Z}$ to $\mathcal{F}_{\text{BC}}$ and receives the output bit b^{out}.
3. It sends $(\text{GETINFO}, R)$ to $\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{\mathsf{median}}}$ and receives the neighborhoods of corrupted parties.
4. Now $\mathcal{S}_{cycle}$ has to simulate the view of all parties in $\mathcal{Z}$. The messages sent by corrupted parties can be easily generated by executing the protocol CycleProt. To simulate the messages sent by honest parties to their corrupted neighbors, $\mathcal{S}_{cycle}$ proceeds as follows.
5. First, it prepares a set `buffer`, containing all messages which will be sent by the honest parties throughout the simulation (recall the variable `buffer` in $\mathcal{F}_{\text{NET}}$). $\mathcal{S}_{cycle}$ initializes $\texttt{buffer} = \varnothing$.
6. $\mathcal{S}_{cycle}$ generates the messages per a connected corrupted arc $P^1, P^2, \ldots, P^K$ of the cycle. We will use the following notation:
   - P^0 and P^{K+1}: the neighboring honest parties.
   - $P^{K+2}, \ldots, P^{n-1}$: (the labels of) the rest of the parties on the cycle (their identities are unknown to $\mathcal{S}_{cycle}$).
   - MedRSum[D]: the distribution corresponding to the median-round-sum of all delays, obtained from $\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{\mathsf{median}}}$.
   - for $0 \leq k \leq n-1$, denote by D_k the delay distribution on the edge from $P^{(k-1) \mod n}$ to P^k (for $1 \leq k \leq K$, D_k was obtained from $\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{\mathsf{median}}}$).
   - for $1 \leq k \leq K$, define $\mathsf{w}^k = \kappa + \left\lceil \text{Med}[D_{(k,k+1)}]/R \right\rceil$ (recall the initialization step of CycleProt).
   - for $1 \leq k \leq K$, denote by $\texttt{pk}^k$ the public key corresponding to the secret keys of the corrupted parties $P^1, \ldots, P^k$.
   - $\texttt{pk}_{sim}$: a public key freshly sampled by $\mathcal{S}_{cycle}$ at the beginning of the simulation.

   $\mathcal{S}_{cycle}$ has to compute the messages sent by P^0 to P^1 and by P^{K+1} to P^K. To compute the former messages, it does as follows (the latter messages are computed analogously):

   1: For $1 \leq k \leq K$, compute the time after which P^0 starts processing messages from the walk started by P^k as $t^k = R(n\kappa + \text{MedRSum[D]} - (\mathsf{w}^1 + \cdots + \mathsf{w}^k))$.
   2: Let $t^0 = R\,(n\kappa + \text{MedRSum[D]})$.
   3: **for** $\tau = t^0$ to 0 and $\tau \equiv 0 \mod R$ **do**
   4: **if** $\tau < t^K$ **then**
   5: Compute $c = \mathsf{PKCR}^*.\mathsf{Enc}(0, \texttt{pk}_{sim})$.
   6: **else**
   7: Find k such that $t^{k+1} \leq \tau < t^k$.

8: Compute $c = \mathsf{PKCR*.Enc}(b^{out}, \mathtt{pk}^k)$.
9: **for** $i = 0$ to $\kappa - 1$ **do**
10: Sample d from D_0.
11: Record the tuple $(\tau + iR + d, P^0, P^1, (\tau/R + i, c))$ in `buffer`.

7. $\mathcal{S}_{cycle}$ simulates the messages received by corrupted parties from $\mathcal{F}_{\text{NET}}$ by executing the algorithm of $\mathcal{F}_{\text{NET}}$. On every input $(\text{FETCHMESSAGES}, j)$ from a corrupted P_j, it gets the current time τ from $\mathcal{F}_{\text{CLOCK}}$. Then, for each message tuple (t, P_i, P_j, c) from `buffer` where $t \leq \tau$, it removes the tuple from `buffer` and outputs (i, c) to P_j.

We first prove a fact about the protocol CycleProt: with overwhelming probability one of the κ copies of a message generated by P_i for P^ℓ in a given round is delivered within w^ℓ rounds.

Lemma 8. *In the real execution of the protocol* CycleProt*, the probability that none of the κ messages (r_c, c) sent by P_i to P^ℓ for round r_c is delivered by round $\mathsf{r}_c + \kappa + \left\lceil \text{Med}[D_{(i,\ell)}]/R \right\rceil$ is negligible.*

Proof. For the distribution $D_{(i,\ell)}$ on the edge between P_i and P^ℓ and for $1 \leq j \leq \kappa$, let X_j be the indicator variable that message (r_c, c) arrived after time $T = R(\mathsf{r}_c + \kappa + \left\lceil \text{Med}[D_{(i,\ell)}]/R \right\rceil)$. Since the message was sent at time $t^j_{sent} = R(\mathsf{r}_c + j)$, with probability $1/2$ the message arrives at P_i by time $t^j_{sent} + \text{Med}[D_{(i,\ell)}]$, and is officially delivered at the next round, time $t^j_{sent} + \left\lceil \text{Med}[D_{(i,\ell)}]/R \right\rceil \cdot R$. Note that for every j, time T is greater than or equal to $t^j_{sent} + \left\lceil \text{Med}[D_{(i,\ell)}]/R \right\rceil \cdot R$. Therefore, for every j, we have

$$\Pr_{D_{(i,\ell)}}[X_j = 1] \geq \frac{1}{2}.$$

Finally, given independence between messages sent at different rounds, the probability that all messages arrive *after* time T is upper bounded by

$$\Pr_{D_{(i,\ell)}}[\sum_{j=1}^{\kappa} X_j = 0] = \prod_{j=1}^{\kappa} \Pr_{D_{(i,\ell)}}[X_j \neq 1] \leq \frac{1}{2^\kappa} = \text{negl}(\kappa).$$

As a consequence, a message sent at round r_c arrives with all but negligible probability at round $\mathsf{r}_c + (\kappa + \left\lceil \text{Med}[D_{(i,\ell)}]/R \right\rceil)$. ■ □

We are now ready to prove that the messages from the walks initiated by P^k are acknowledged and processed by P^0 after exactly t^k clock ticks. This also proves correctness of the protocol: P^0 will get his walk back after t^0 clock ticks.

Lemma 9. *In the real execution of the protocol, at time t^k, the party P^0 starts sending to P^1 a message from the walk started by P^k.*

Proof. We assume that for each P^i, one of the copies of a message generated by P^{i-1} in a given round arrives within w^i rounds. By Lemma 8, this happens with overwhelming probability.

Consider each message-walk started by party P^k going to P^{k+1} (we ignore the repetitions and count the number of distinct messages sent). There are w^k such walks, started at rounds $0, \ldots, \mathsf{w}^k - 1$. A walk started at a given round is processed by the party $P^{k+k'}$ after $\sum_{i=1}^{k'} \mathsf{w}^{k+i}$ rounds. So it is processed by P^0 after $\sum_{i=0}^{n-1} \mathsf{w}^i - (\mathsf{w}^1 + \cdots + \mathsf{w}^k)$ rounds. We have $\sum_{i=0}^{n-1} \mathsf{w}^i = \text{MedRSum[D]}$. Hence, P^0 processes messages from the walk started by P^k after time $R \cdot (\text{MedRSum[D]} - \sum_{i=1}^{k} \mathsf{w}^i) = t^k$. ■ □

Remark 3. *The protocol is correct. Note that Lemma 9 implies the correctness of the protocol. A party "sends" a message first by processing it (and seeing if it can decrypt) and then forwarding it if it did not decrypt. Without loss of generality, consider party P_1. The walk started from P_1 arrives back at P_1 at time t^n, at which point, $n-1$ layers of encryption will have been removed, and the message is guaranteed to have passed the source party (and thus have the output bit). P_1 decrypts this message and gets the output from the protocol.*

Finally, we show that the execution with $\mathcal{S}_{cycle}$ is indistinguishable from the real execution by presenting a sequence of hybrids. In the following, we only consider the messages sent by an honest $P_i = P^0$ to its corrupted neighbor $P_j = P^1$ (all other messages are trivial to simulate).

Hybrid 1. *$\mathcal{S}^1_{cycle}$ simulates the real world exactly. That is, $\mathcal{S}^1_{cycle}$ has information on the entire communication graph and all edge delays. It generates messages according to the protocol, at the time they are sent.*

Hybrid 2. *$\mathcal{S}^2_{cycle}$ generates all messages upfront the same way $\mathcal{S}_{cycle}$ does, but the messages are still generated according to the protocol.*

Hybrid 3. *$\mathcal{S}^3_{cycle}$ replaces the real ciphertexts $\mathsf{PKCR*.DelLayer}(c, \mathtt{sk}_i)$ sent by P^0 by fresh encryptions $\mathsf{PKCR*.Enc}(m, \mathtt{pk}')$ (where m is the message c encrypts, and $\mathtt{pk}'$ is the public key corresponding to the secret keys of the parties remaining on the cycle).*

Hybrid 4. *For messages generated at time $\tau < t^K$, $\mathcal{S}^4_{cycle}$ changes the encryption key to $\mathtt{pk}_{sim}$, that is, it computes $\mathsf{PKCR*.Enc}(m, \mathtt{pk}_{sim})$ instead of $\mathsf{PKCR*.Enc}(m, \mathtt{pk}')$. For messages generated at time $\tau \geq t^K$, $\mathcal{S}^4_{cycle}$ uses fresh encryptions under the public key $\mathsf{PKCR*.Enc}(m, \mathtt{pk}^k)$, where k is chosen as in $\mathcal{S}_{cycle}$.*

Hybrid 5. *For messages generated at time $\tau < t^K$, $\mathcal{S}^5_{cycle}$ changes the message to 0, i.e., it computes $\mathsf{PKCR*.Enc}(0, \mathtt{pk}_{sim})$ instead of $\mathsf{PKCR*.Enc}(m, \mathtt{pk}_{sim})$.*

Hybrid 6. *For messages generated at time $\tau \geq t^K$, $\mathcal{S}^6_{cycle}$ computes the encrypted message using the output of $\mathcal{F}_{\text{BC}}$.*

Observe that Hybrids 1 and 2 are trivially identical. Moreover, indistinguishability of Hybrids 5 and 6 follows from the correctness of the protocol, allowing the adversary to decrypt not the bit generated by traversing the graph, but the bit generated by the ideal functionality. These two will be equivalent since each message traverses the entire graph. It is also easy to see that $\mathcal{S}^6_{cycle}$ is the original simulator $\mathcal{S}_{cycle}$.

Claim 8. *No efficient distinguisher can distinguish between Hybrids 2 and 3.*

Proof. For an honest party P_i, $\mathcal{S}^3_{cycle}$ generates all messages sent by it as fresh encryptions, while a message generated by $\mathcal{S}^2_{cycle}$ can be one of the following:

- An initial ciphertext (starting the cycle): this is the same as in $\mathcal{S}^3_{cycle}$.
- A ciphertext c, which results from applying to another ciphertext c', in order, PKCR*.DelLayer, and PKCR*.Rand. By correctness, c is an encryption of the same message as c' with overwhelming probability. Moreover, re-randomizability of PKCR*, guarantees that the distribution of c is indistinguishable from the distribution of a fresh encryption of the same message, as generated by $\mathcal{S}^3_{cycle}$.
- A ciphertext c, which results from applying to another ciphertext c', in order, PKCR*.DelLayer, PKCR*.ToOne, and PKCR*.Rand. As in the previous case, c is an encryption of one with overwhelming probability, and the distribution of c is indistinguishable from the distribution of a fresh encryption of 1.

■

Claim 9. *No efficient distinguisher can distinguish between Hybrids 3 and 4.*

Proof. We will prove first that messages sent after time t^K are indistinguishable between the two hybrids, and then show that before time t^K, they are also indistinguishable.

After t^K: Consider each message walk started by corrupted party P^k going to P^{k+1}. By Lemma 9, by time t^k, in the real world, the messages from that walk would have traversed all parties except $P_1, \ldots, P_{k-1}$. So, by that time, all key-layers except those from corrupted parties $P_1, \ldots, P_{k-1}$ will have been removed, meaning that message is now encrypted under public key pk^k. Hybrids 3 and 4 are equivalent in this case, then, because we are actually encrypting under the same key as one would encrypt in the real world.

Before t^K: By Lemma 9, the messages sent by P^0 before t^K are encrypted under the public key, for which the honest party P^{K+1} holds a part of the secret key. So, the proof for time before t^K is a reduction to a version of KI-CPA security of PKCR*, where the adversary is allowed to ask many encryption queries. We note that [19] defines only the game for one query, but the reduction follows by a standard hybrid argument. For completeness, we recall their game against an adversary $\mathcal{A}$:

Game IK-CPA

1: $\mathtt{pk}_0, \mathtt{sk}_0 = \mathsf{PKCR*.KGen}(1^\kappa)$
2: $\mathtt{pk}_1, \mathtt{sk}_1 = \mathsf{PKCR*.KGen}(1^\kappa)$
3: Choose $b \in \{0,1\}$ at random.
4: $b' = \mathcal{A}^{\mathsf{PKCR*.Enc}(\cdot,\mathtt{pk}_b)}(\mathtt{pk}_0, \mathtt{pk}_1)$
5: Output $b = b'$.

Recall that, for each honest party P_i, $\mathcal{S}^3_{cycle}$ uses the real key corresponding to a number of parties on the cycle, while $\mathcal{S}^4_{cycle}$ uses a different key $\mathtt{pk}_{sim}$. We will introduce a sequence of intermediate hybrids H_1 to H_n, where H_i uses the real key for $P_1, \ldots, P_i$, and the simulated key for the other parties (in case they are honest and have corrupted neighbors).

Assume that $\mathcal{D}$ is a distinguisher for Hybrids H_{i-1} and H_i. A winner $\mathcal{A}$ for the above game can be constructed as follows. Let $\mathtt{pk}_0, \mathtt{pk}_1$ be the keys obtained from the game. If $\mathcal{D}$ corrupts P_i or it does not corrupt any of its neighbors, we abort (the hybrids are trivially indistinguishable). Otherwise, let $P^1, \ldots, P^K$ be the corrupted arc starting at P_i's neighbor P^1. $\mathcal{A}$ will simulate the protocol for $\mathcal{D}$ using freshly generated key pairs, except that for P^{K+1} it will use its challenge key $\mathtt{pk}_0$. Note that it can now efficiently compute the joint key. Ciphertexts sent by the parties can now be generated using $\mathcal{A}$'s encryption oracle. $\mathcal{A}$ outputs whatever $\mathcal{D}$ outputs.

■

Claim 10. *No efficient distinguisher can distinguish between Hybrids 4 and 5.*

Proof. In both Hybrid 4 and Hybrid 5, the messages sent before D_k are encrypted under a fresh public key $\mathtt{pk}_{sim}$ chosen by the simulator. Hence, the indistinguishability follows by semantic security. ■

2.13 Details of the Protocol for Trees

Protocol TreeProt

// The common input of all parties is the round length R. Additionally, the sender P_s has the input bit b_s.

Setup: *For $i \in \{1, \ldots, n\}$, let $(\mathtt{pk}_i, \mathtt{sk}_i) = \mathsf{PKCR*.KGen}(1^\kappa)$. Let $\mathtt{pk} = \mathtt{pk}_1 \circledast \ldots \circledast \mathtt{pk}_n$.*

The setup outputs to each party P_i its secret key $\mathtt{sk}_i$ and the product public key $\mathtt{pk}$.

Initialization for each P_i:

- *Send* (GetInfo) *to the functionality $\mathcal{F}_{\text{NET}}$ to obtain the distributions of the local edges.*
- *Assign randomly the labels $P^0, \ldots, P^{\nu-1}$ to its neighbors. Let $\mathtt{succ}(\ell) = \ell + 1 \mod \nu$ denote the index of the successor of neighbor P^ℓ on the tree traversal.*
- *For each neighbor P^ℓ, initialize a list $\mathsf{Rec}^\ell = \perp$ and a set $\mathsf{Send}^\ell = \varnothing$.*
- *For each ℓ, $D_{(i,\ell)}$ is the delay distribution on the edge between P_i and neighbor*

P^ℓ, *obtained from* $\mathcal{F}_{\text{INFO}}$.

- *Let* $\mathsf{w}^\ell = \kappa + \left\lceil \text{Med}[D_{(i,\ell)}]/R \right\rceil$ *be the time* P_i *waits before sending a message from* P^ℓ *to* $P^{\ell+1}$.

Execution for each P_i***:***

1: *Send* (ReadClock) *to the functionality* $\mathcal{F}_{\text{CLOCK}}$ *and let* τ *be the output. If* $\tau \bmod R \neq 0$, *send* (Ready) *to the functionality* $\mathcal{F}_{\text{CLOCK}}$. *Otherwise, let* $\mathsf{r} = \tau/R$ *be the current round number and do the following:*

2: Receive messages: *Send* (FetchMessages, i) *to the functionality* $\mathcal{F}_{\text{NET}}$. *For each message* (r_c, c) *received from a neighbor* P^ℓ, *set* $\mathsf{Rec}^\ell[\mathsf{r}_c + \mathsf{w}^\ell] = c$.

3: Process messages from P^ℓ with $\ell \neq 0$:

- *For each* $P^\ell \neq P^0$ *such that* $\mathsf{Rec}^\ell[\mathsf{r}] = \perp$, *add* $(\kappa, \mathsf{r}, \mathit{PKCR*.Enc}(0, \mathit{pk}))$ *to* $\mathsf{Send}^{\mathtt{succ}(\ell)}$.
- *For each* $P^\ell \neq P^0$ *such that* $\mathsf{Rec}^\ell[\mathsf{r}] \neq \perp$, *add* $(\kappa, \mathsf{r}, \mathit{PKCR*.Rand}(\mathsf{Rec}^\ell[\mathsf{r}]))$ *to* $\mathsf{Send}^{\mathtt{succ}(\ell)}$.

4: Process if no messages received from P^0: *If* $\mathsf{Rec}^0[\mathsf{r}] = \perp$, *start a new cycle traversal in the direction of* P^1*:*

- *If* P_i *is sender (i.e.* $i = s$*) then add* $(\kappa, \mathsf{r}, \mathit{PKCR*.Enc}(b_s, \mathit{pk}))$ *to* Send^1.
- *Otherwise, add* $(\kappa, \mathsf{r}, \mathit{PKCR*.Enc}(0, \mathit{pk}))$ *to* Send^1.

5: Process received messages from P^0: *Set* $\mathsf{d} = \mathit{PKCR*.DelLayer}(\mathsf{Rec}^0[\mathsf{r}], \mathit{sk}_i)$, *and do the following:*

- *If* $\mathsf{d} \in \{0, 1\}$, *output* d *and halt (we have decrypted the source bit).*
- *Otherwise, if* $i = s$ *and* $b_s = 1$, *then set* $\mathsf{d} = \mathit{PKCR*.ToOne}(\mathsf{d})$. *Then, in either case, add* $(\kappa, \mathsf{r}, \mathit{PKCR*.Rand}(\mathsf{d}))$ *to* Send^1.

6: Send messages: *For each* $\ell \in \{0, \ldots, \nu - 1\}$, *let* $\mathsf{Sending}^\ell = \{(k, \mathsf{r}_c, c) \in \mathsf{Send}^\ell : k > 0\}$. *For each* $(k, \mathsf{r}_c, c) \in \mathsf{Sending}^\ell$, *send* (r_c, c) *to* P^ℓ.

7: Update Send sets: *For each* $(k, \mathsf{r}_c, c) \in \mathsf{Sending}^\ell$, *remove* (k, r_c, c) *from* Send^ℓ *and insert* $(k - 1, \mathsf{r}_c, c)$ *to* Send^ℓ.

8: *Send* (Ready) *to the functionality* $\mathcal{F}_{\text{CLOCK}}$.

2.13.1 Protocol for General Graphs

2.14 Protocol for General Graphs

We present a protocol that allows us to securely realize any functionality in any connected communication graph with unknown delay distributions on the edges. For that, we use the same setup as [14]: we assume that the parties have access to secure hardware boxes, initialized with the same secret key, and executing the same functionality $\mathcal{F}_{\text{HW}}$, *independent of the graph and the realized functionality (see [14] for details of this model).*

Our protocol is divided into two sub-protocols: preprocessing and computation. Both sub-protocols do not terminate on their own. Rather, we assume that each party gets a signal when it can finish each sub-protocol.[14] *The preprocessing is executed only once, before any input is specified and can be re-used. Intuitively, it outputs, for each party, an encryption of the entire communication graph under the secret key embedded in the hardware boxes. The computation allows to evaluate any function, with the help of the encrypted information outputted by the preprocessing. One output of preprocessing can be used to execute the computation any number of times, each time with different function and different inputs.*

In the following, we formally describe both protocols. To make the exposition easier to follow, we postpone the precise definition of the functionality $\mathcal{F}_{\text{HW}}$ executed by the hardware boxes, to Appendix 2.15, and for now only give an informal description of its behavior whenever $\mathcal{F}_{\text{HW}}$ is invoked.

2.14.1 Preprocessing

The preprocessing is executed without any inputs. The output is a pair $(\mathtt{id}_i, c)$, where $\mathtt{id}_i$ is a (secret) random string used to identify a party, and c is a ciphertext that contains an encrypted state with the whole graph. This output pair will be inputted to the computation protocol.

At a high level, the protocol floods the network with encrypted partial images of the graph, until the signal to terminate occurs. We assume that the signal occurs late enough for all parties to collect all information. In more detail, throughout the protocol, a party P_i keeps an encrypted state c, containing information about the graph and parties' $\mathtt{id}$'s, that it collected up to a given point. Initially, c contains only the local neighborhood and $\mathtt{id}_i$ chosen at random by P_i. Then, every round, P_i sends c to all its neighbors. When it receives a state c_j from a neighbor P_j, it uses the functionality $\mathcal{F}_{\text{HW}}$ box to update c with the information from c_j. That is, $\mathcal{F}_{\text{HW}}$ gets as input two encrypted states containing partial images on the graph, respectively, decrypts both states and merges the information into a new state, which is encrypted and output.

Protocol Hw-Preprocessing

// The common input of all parties is the round length R.

Setup: *Each party P_i has access to a secure hardware box functionality $\mathcal{F}_{\text{HW}}$.*

Initialization for each P_i: *Choose an identifier $\mathtt{id}_i$ at random and send* (GetInfo) *to $\mathcal{F}_{\text{NET}}$, to obtain the neighborhood $\mathbf{N}_G(P_i)$. Input $(i, \mathtt{id}_i, \mathbf{N}_G(P_i))$ to $\mathcal{F}_{\text{HW}}$ and store the resulting encrypted state c.*

[14] In practice, this is not an unrealistic assumption. It would be enough, for example, if each party was given a very rough upper bound on the time it takes to flood the network and traverse all edges of the graph (for instance, a constant number proportional to the sum of delays on all edges). This is still faster than assuming worst-case upper bounds on the delays along edges, as one would need to do to adapt a fully synchronous protocol.

Execution for each P_i at every round (every R clock ticks):

1: *Send c to each $P_j \in \mathbf{N}_G(i)$.*
2: *Send* $(\text{FETCHMESSAGES}, i)$ *to* $\mathcal{F}_{\text{NET}}$. *For each received message c', input* $(\mathtt{id}_i, c, c')$ *to* $\mathcal{F}_{\text{HW}}$ *and set the updated state c to the result.*

Termination for each P_i: *Upon receiving the signal, output* $(\mathtt{id}_i, c)$.

2.14.2 Computation

The inputs to the computation protocol are, for every P_i, its input x_i, a description of the function f_i that evaluates P_i's output of the computed function, and the values $\mathtt{id}_i$ and c_i, outputted by preprocessing.

The high-level idea is that the hardware box $\mathcal{F}_{\text{HW}}$ gets as part of its input the value c_i, containing, among others, the encrypted communication graph. This allows it to deterministically compute an Eulerian cycle, which visits every edge exactly twice. Then, every party starts a traversal of the Eulerian cycle, in order to collect the inputs from all parties. Once all inputs are collected, the box computes the function and gives the output to the party. Traversing each edge exactly twice allows all parties to learn the output at a time that does not depend on the graph topology but (roughly) on the distribution of the sum of the delays. Of course, all messages are encrypted under the secret key embedded in the hardware boxes.

This means that at any time during the protocol there are n cycle traversals going through the graph (one per a starting party). Each of the traversals visits all edges in the graph twice. So in each round a party P_i processes messages for up to n traversals. To hide the number of actual traversal processed P_i sends n messages to each each of its neighbors. This means that each round, P_i receives from each neighbor n messages. It inputs all of them to its hardware box (together with its input to the computed function) and receives back, for each neighbor, a set of n messages that it then sends to him.

A party receives the output once the cycle has been traversed, which takes time proportional to the sum of the rounded delays. Once the parties receive output, they continue executing the protocol until they receive the termination signal, which we assume occurs late enough for all *parties to get their outputs.*

There are still some subtle issues, that the above sketch does not address. First, the adversary could try to tamper with the ciphertexts. For example, in our protocol a message contains a list of $\mathtt{id}$'s that identifies the path it already traversed. This is done so that the adversary cannot extend the traversal on behalf of an honest party P_i without knowing its secret $\mathtt{id}_i$. Now the adversary could try to extend this list nevertheless, by copying part of the encrypted state of a corrupted party — recall that this state contains all $\mathtt{id}_i$'s. To prevent such situations, we use authenticated encryption.

Second, we need to specify when the parties input the function they are evaluating into the box. Doing this at the very end would allow the adversary to evaluate many

functions of her choice, including the identity. So instead, in our protocol the function is inputted once, when the cycle traversal is started, and it is always a part of the message. This way, when the output is computed, the function is taken from a message that has been already processed by all honest parties. Since honest parties only process messages that are actually sent to them, and even corrupted parties only send correctly generated messages, this function must be the correct one. In some sense, when sending the first message to an honest party, the adversary commits herself to the correct function.

A similar problem occurs when the parties input to their boxes the inputs to the computed function. A sequence of corrupted parties at the end of the traversal can emulate the last steps of the protocol many times, with different inputs. To prevent this, we traverse the cycle twice. After the first traversal, the inputs are collected and the function is evaluated. Then, the (still encrypted) output traverses the cycle for the second time, and only then is given to the parties.

Finally, we observe that at the end of the protocol, a graph component of neighboring corrupted parties learns where the traversal enters their component (this can be done by fast-forwarding the protocol). Depending on how the eulerian cycle is computed, this could leak information about the topology. To address this, we introduce in Section 2.14.3 an algorithm for computing the traversal that does not have this issue (formally, the last part of the cycle can be simulated).

Protocol Hw-Computation

// The common input of all parties is the round length R. Additionally, each P_i has input $(x_i, f_i, \mathtt{id}_i, c_i)$, where $\mathtt{id}_i$ is the identifier chosen in Hw-Preprocessing, and c_i is the encrypted state outputted by Hw-Preprocessing.

Setup: *Each party P_i has access to a secure hardware box functionality $\mathcal{F}_{\text{HW}}$.*

Initialization for each P_i: *For each neighbor P_j, let $E_j = \varnothing$.*

Execution for each P_i at every r clock ticks:

1: *Send* (FetchMessages) *to $\mathcal{F}_{\text{NET}}$ and receive the messages $(E_1, \ldots, E_\nu)$.*
2: *Choose r at random and input $(i, \mathtt{id}_i, c_i, \bigcup_j E_j, x_i, f_i, r)$ to $\mathcal{F}_{\text{HW}}$. Get the result $(\mathtt{val}, \{(E'_1, \mathtt{next}_1), \ldots, (E'_k, \mathtt{next}_k)\})$. If $\mathtt{val} \neq \perp$, output $\mathtt{val}$, but continue running.*
3: *For each $(E'_j, \mathtt{next}_j)$, for each $e \in E'_j$, send e to $\mathtt{next}_j$ via* (Send, $i, \mathtt{next}_j, e$).[15]

Termination for each P_i: *Upon receiving the signal, terminate.*

Realizing reactive functionalities. *Reactive functionalities are those which require explicit interaction between parties, e.g. if the function we realize is very simple but*

[15]We will assume that every message sent in this round is independent. In this case this is equivalent to assuming only independence between rounds — since there is an upper bound n on the number of messages sent at once, one can always make the round longer, partition it into slots separated by a sufficient time interval, and send one message in every slot.

we want to evaluate a complex function, parties may need to run this protocol multiple times in sequence, using previous outputs to generate the next inputs. Our current hardware protocol allows us to realize secure function evaluation. In the synchronous setting, this can be easily extended to reactive functionalities by invoking many function evaluations in sequence. However, in the setting with unknown delays this is no longer clear. For example, if our protocol is composed sequentially in the naive way, then parties start the second execution at different times, which leaks topology.

So, to get reactive functionalities or composition to work for this hardware protocol we can do one of two things. First, we could add a synchronization point before each 'round' of the reactive function. Second, we could employ the same trick as for the cycle/tree protocol in Section 2.11.1, sending the same message many times so that with high probability it arrives to the next node within some reasonable time interval. With this method, every party ends the protocol at exactly the same time, and so can start the next protocol at the same time, despite the delays.

The running time of the protocol Hardware *depends only on the sum of all delays in the network, each rounded to the next multiple of the round length R, which is the only information leaked in the ideal world. In Appendix 2.16 we prove the following theorem.*

Theorem 9. *For any efficiently computable and well-formed*[16] *functionality $\mathcal{F}$, the protocol* Hardware *UC-realizes $(\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{INFO}}^{\mathcal{L}_{\mathsf{sum}}}, \mathcal{F})$ in the $(\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{NET}}, \mathcal{F}_{\text{HW}})$-hybrid model with an adversary who statically passively corrupts any number of parties, where $\mathcal{L}_{\mathsf{sum}} \coloneqq R \sum_{D_e \in \mathcal{D}} \lceil D_e/R \rceil$.*

Remark. *One can observe that in our protocol the hardware boxes must be able to evaluate a complex function. This can be resolved at the cost of efficiency, by computing the functionality by many calls to the simple broadcast functionality. Note that even if we require one synchronization point per broadcast, this still seems reasonable, since it is possible to evaluate any function with constant number of broadcasts [50, 99].*

2.14.3 Computing the Eulerian Cycle

It turns out that not every algorithm computing an Eulerian cycle can be used in $\mathcal{F}_{\text{HW}}$ to achieve THC. In particular, during the execution of our protocol the adversary learns some information about a part of the cycle, which for some algorithms depends on the graph. More technically, during the simulation, it is necessary to compute the time when the adversary learns the output, and this happens as soon as the Eulerian cycle traversal enters a fragment of consecutive corrupted parties containing the output party. This is because it can "fast-forward" the protocol (without communication). Hence, we need an algorithm for computing such a cycle on a graph with doubled edges, for which the "entry point" to a connected component (of corrupted parties) can be simulated with only the knowledge of the component.

[16] Intuitively, a functionality is well-formed if its code does not depend on the ID's of the corrupted parties. We refer to [33] for a detailed description.

Common algorithms, such as Fleury or Hierholzer [58, 57], check a global property of the graph and hence cannot be used without the knowledge of the entire graph topology. Moreover, a distributed algorithm in the local model (where the parties only have knowledge of its neighbors) such as [103] is also not enough, since the algorithm has to be executed until the end in order to know what is the last part of the cycle.

We present the algorithm EulerianCycle, which, if executed from a node u on a connected neighborhood containing u, leads to the same starting path as if it was executed on the whole graph. This property is enough to simulate, since the simulator can compute the last fragment of the Eulerian Cycle in the corrupted neighborhood. We note that the start of the cycle generated by our algorithm can be simulated, however, the simulator needs to compute the end. Hence, the hardware boxes will traverse the path outputted by EulerianCycle from the end.

The idea is to generate a tree from the graph, in such a way that the generated tree contains exactly the same edges as the graph. To do that, the tree is generated in a DFS-manner from a source u. At every step, a new edge (the one that leads to the smallest id according to a DFS order, and without repeating nodes) is added to the tree. Since the graph is connected, all edges are eventually added. Moreover, each edge is added exactly once, since no repeated nodes are expanded. See Figure 2-3 for an example execution.

Algorithm EulerianCycle$(u, G = (E, V))$

// Computes an eulerian cycle on the graph G with the set of nodes V and the set of edges E (where each edge is considered doubled), starting at node $u \in V$. We assume some ordering on V.

Let $\mathcal{T}$ be the tree with a single root node u.
while *$E \neq \varnothing$* ***do***
    if *there is no $v \in V$ such that $(u, v) \in E$* ***then***
        Set $u = \mathsf{parent}(\mathcal{T}, u)$
    else
        Pick the smallest v such that $(u, v) \in E$ and append v to the children of u in $\mathcal{T}$.
        Set $E = E \setminus \{(u, v)\}$.
        If $v \notin \mathsf{nodes}(\mathcal{T})$, then set $u = v$.
Output the path corresponding to the in-order traversal of $\mathcal{T}$.

2.15 The Function Executed by the Hardware Boxes

The functionality $\mathcal{F}_{\text{HW}}$ contains hard-wired the following values: a symmetric encryption key pk, and a key rk for a pseudo-random function prf. Whenever it outputs an encryption, it uses an authenticated encryption scheme AE with key pk, and with encryption randomness computed as $\mathsf{prf}_{rk}(x)$, where x is the whole input of $\mathcal{F}_{\text{HW}}$. $\mathcal{F}_{\text{HW}}$ can receive three types of input, depending on the current stage of the protocol: the initial input and an intermediate input during $\mathsf{Hw\text{-}Preprocessing}$, and an intermediate

Figure 2-3: An example of a graph G (on the left) and the corresponding tree $\mathcal{T}$, computed by $\mathsf{EulerianCycle}(1, G)$ (on the right). The eulerian cycle (on the graph with doubled edges) is $(1, 2, 4, 1, 3, 1, 3, 5, 3, 4, 2, 1)$.

input during Hw-Computation. On any other inputs, $\mathcal{F}_{\text{HW}}$ outputs $\perp$.

Behavior during preprocessing. *During the preprocessing, the first input is a triple $(i, \mathtt{id}_i, \mathbf{N}_G(P_i))$, and next inputs are triples $(\mathtt{id}, c, c_j)$, where c and c_j are states of parties, encrypted under pk. In particular, the state of a party P_i consists of the following information:*

- *i: the index of P_i,*
- *G: the current image of the graph (stored in an n-by-n matrix),*
- *$\mathtt{ID} = (\mathtt{id}_1, \ldots, \mathtt{id}_n)$: a vector, containing the currently known identifiers of parties.*

On the first input, $\mathcal{F}_{\text{HW}}$ outputs an encryption of the initial state, that is, the state where the graph G contains only the direct neighborhood of P_i, and $\mathtt{ID}$ contains only the value $\mathtt{id}_i$ chosen by P_i. For the inputs of the form $(\mathtt{id}, c, c_j)$, $\mathcal{F}_{\text{HW}}$ decrypts the states c and c_j and merges the information they contain into a new state s, which it then encrypts and outputs.

Behavior during computation. *Recall that the goal of $\mathcal{F}_{\text{HW}}$ at this stage is to compute the next encrypted messages, which a party P_i will send to its neighbors. That is, it takes as input a set of encrypted messages received by P_i and, for each neighbor of P_i, outputs a set of n messages to be sent.*

Each encrypted message contains information about which graph traversal it is a part of, about the current progress of the traversal, and about all the inputs collected so far. Moreover, we include the information from the encrypted state: $(i, G, \mathtt{ID})$ and the function f of the party starting the cycle. Intuitively, the reason for including f and the encrypted state is that, since the adversary is passive, the information taken from the message must be correct (for example, now a corrupted party cannot use its box to evaluate any function of its choice). Formally, an encrypted message from another node decrypts to a message m_j containing the following elements:

- *j is the party number (the publicly known number between 1 and n, not the party's* $\mathtt{id}$*)*
- $\mathtt{ID}_j$ *is the vector of unique random id's. Carrying this in the message allows us to ensure that inputs are all consistent with the same parties.*
- G_j *is the adjacency matrix of the network graph. It is also used to check consistency.*
- $\mathtt{Path}_j = (\mathtt{id}^1, \ldots, \mathtt{id}^{4n^2})$*: a vector of length $4n^2$, containing the current set of identifiers of parties visited so far along the graph traversal starting at P_j (recall that the eulerian cycle of length at most $2n^2$ is traversed twice).*
- f_j *is the function that parties will compute.*
- $\mathbf{x}_j$ *is a vector that has a slot for every party to put its input. It starts as being completely empty, but gains an entry when it visits a new node on the graph. We also check this for consistency (a party trying to input a different value from the one they started with will not be able to use the hardware).*

At a high level, $\mathcal{F}_{\text{HW}}$ first discards any dummy or repeated messages (a party can receive many messages, but the hardware box needs to continue at most n Eulerian cycles), and then processes each remaining message. If a message has traversed the whole Eulerian cycle, $\mathcal{F}_{\text{HW}}$ computes and reveals the function applied to the inputs. Otherwise, it creates an encryption of a new message with the current party's id added to the current path, and its input added to the list of inputs, and $\mathtt{next}$ *contains the id of the destination neighbor. After processing all messages, for each destination neighbor, it adds correctly formated dummy encryptions, so that exactly n encryptions are sent to each neighbor.*

The functionality $\mathcal{F}_{\text{HW}}$ is formally described below. It calls the following subroutines:

- **AggregateTours** *takes as input a set of messages M. Each of these messages contain information about a Eulerian Cycle, the party that started that Eulerian Cycle, and the path traversed so far. The subroutine selects the (at most n) messages that start from different parties. It is expected that Eulerian Cycles starting from the same party, are exactly the same message.*
- **ContinueTour** *takes as input a specific message, a Eulerian Cycle that the message must traverse, and a current party's input and number. If the Eulerian Cycle has not been traversed, it then creates a new message containing a path with the current party's input and id appended to the corresponding variables, and also the id of the party where the message should be sent. Otherwise, it outputs a flag indicating that the Eulerian Cycle has ended and the output must be revealed.*
- **EncryptAndFormatOutput** *takes as input a set of pairs message-dest- ination, and appends to each possible destination parsable messages until there are*

n messages. It then encrypts each message and outputs, for each possible destination a set of encryptions and the id of the party where the encryptions must be sent.

Functionality $\mathcal{F}_{\mathrm{HW}}$

Setup: *The hardware box is initialized with a symmetric encryption key pk and a PRF key rk.*

Initial input during Hw-Preprocessing

Input: $x = (i, \mathtt{id}_i, \mathbf{N}_G(P_i))$

Compute the initial vector $\mathtt{ID}$ *as a vector of* n $\perp$*'s except with* $\mathtt{id}_i$ *in the* i*-th position.*
Compute a new adjacency matrix G_i *with the only entries being the local neighborhood of* P_i*.*
Compute the initial state $s = (i, \mathtt{ID}, G_i)$

Output: *the encrypted initial state* $\mathsf{AE.Enc}_{pk}(s; \mathsf{prf}_{rk}(x))$*.*

Intermediate input during Hw-Preprocessing

Input: $x = (\mathtt{id}, c, c_j)$*, where* $\mathtt{id}$ *is the identifier of* P_i*,* c *is the encrypted state of* P_i*, and* c_j *is the state of a neighbor* P_j*.*

Compute the states $(i, \mathtt{ID}, G) = \mathsf{AE.Dec}_{pk}(c)$ *and* $(j, \mathtt{ID}_j, G_j) = \mathsf{AE.Dec}_{pk}(c_j)$*.*
Compute the new state $s = (i, \mathtt{ID}', G')$*, where* $\mathtt{ID}'$ *contains all identifiers which appear in* $\mathtt{ID}_j$ *or* $\mathtt{ID}$*, and* G' *is the union of* G *and* G_j*.*

Output: *the encrypted state* $\mathsf{AE.Enc}_{pk}(s; \mathsf{prf}_{rk}(x))$*.*

Intermediate input during Hw-Computation

Input: $x = (i, \mathtt{id}, c, E, x_i, f_i, r)$*, where* i *is the party's index,* $\mathtt{id}$ *is the identifier of* P_i*,* c *is the encrypted state of* P_i*,* E *is the set of encrypted messages (freshly gotten from the buffer),* x_i *is the input,* f_i *is the evaluated function and* r *is a fresh random value.*

Decrypt the messages $M = \{\mathsf{AE.Dec}_{pk}(e) \mid e \in E\}$ *(output* $\perp$ *if any decryption fails).*
Let $L = AggregateTours(M)$*, and output* $\perp$ *if AggregateTours outputs* $\perp$*.*
Let $S = \varnothing$*,* $\mathtt{val} = \perp$*.*
if $L = \varnothing$ ***then****// Start the traversal.*
    Decrypt the state $(i, \mathtt{ID}, G) = \mathsf{AE.Dec}_{pk}(c)$ *(output* $\perp$ *if the decryption fails). // The graph and the ID-vector are taken from the encrypted state.*
    Let $\mathtt{Path} = (\mathtt{id}, \perp, \ldots, \perp)$ *be a vector of length* $4n^2$*. Let* $\mathtt{x}$ *be the vector of length* n*, initialized to* $\perp$ *and set* $\mathtt{x}[i] = x_i$*.*
    Compute $\mathtt{Tour}_i$ *as the reverse Euler Cycle for* G *starting at party* P_i*.*
    Let $m = (i, \mathtt{ID}, G, \mathtt{Path}, f_i, \mathtt{x})$*.*
    Add $(m, \mathtt{Tour}_i[2])$ *to* S*.*
else*// Continue traversals.*
    for $m \in L$ ***do***

12: *Parse* $m = (j, \texttt{ID}_j, G_j, \texttt{Path}_j, f_j, \mathbf{x}_j)$. *// The graph and the ID-vector are taken from the message.*
13: *Compute* $\texttt{Tour}_j$ *as the reverse Euler Cycle for* G *starting at party* P_j.
14: *Parse* $\texttt{Path}_j = (p_1, \ldots, p_{\ell_j}, \perp, \ldots, \perp)$. *Output* $\perp$ *if any of the following conditions holds:*

- $\texttt{id} \neq \texttt{ID}_j[i]$
- $p_{\ell_j} \neq i$
- *for any* $l \in [\ell_j]$, $p_l \neq \texttt{Tour}_j[l \mod 2m]$

15: *Let* $(m', \texttt{next}) = \mathit{ContinueTour}(m, x_i, i, \texttt{Tour}_j)$.
16: ***if*** $m' = \texttt{Output}$ ***then***
17: *Let* $\texttt{val} = f_i(\mathbf{x}_j)$.
18: ***else***
19: *Add* $(m', \texttt{next})$ *to* S.
20: ***Output :*** *(*$\texttt{val}$*,* $\mathit{EncryptAndFormatOutput}(i, G, r, S, 0)$*)*

Functionality $\mathcal{F}_{\text{HW}}$-subroutines

AggregateTours (M)

// Takes a set of messages and for each party outputs a message that corresponds to its Euler Cycles.

1: *If any* $m \in M$ *does not parse properly, return* $\perp$.
2: *Let* $L = \varnothing$.
3: ***for each*** $m \in M$ ***do***
4: *Parse* $m = (j, \texttt{ID}, G, \texttt{Path}, f, \mathbf{x})$.
5: ***if*** $\exists m' := (j, *, *, *, *, *) \in L$ *and* $m' \neq m$ ***then***
6: *Output* $\perp$.
7: ***if*** $m \notin L$ ***then***
8: *Add* m *to* L.
9: ***return*** L

ContinueTour $(m_j, x_i, i, \texttt{Tour}_j)$

1: *Parse* $m_j = (j, \texttt{ID}_j, G_j, \texttt{Path}_j, f_j, \mathbf{x}_j)$.
2: *Parse* $\texttt{Path}_j = (p_1, \ldots, p_{\ell_j}, \perp, \ldots, \perp)$.
3: ***if*** $\ell_j = 4m - 1$ *and* $\texttt{Tour}_j[(\ell_j + 1) \mod 2m] = i$ ***then***
4: ***return*** $(\texttt{Output}, 0)$.
5: *Set* $\texttt{Path}_j = (p_1, \ldots, p_{\ell_j}, \texttt{Tour}_j[(\ell_j + 1) \mod 2m], \perp, \ldots, \perp)$.
6: *If* $\mathbf{x}_j[i] = \perp$, *then set* $\mathbf{x}_j[i] = x_i$.
7: ***return*** $(m_j, \texttt{Tour}_j[(\ell_j + 1) \mod 2m])$.

EncryptAndFormatOutput $(i, G, r, S, \texttt{sim})$[17]

1: *For each* $d \in \mathbf{N}_G(i)$, *let* $M_d = \{m : (m, d) \in S\}$.
2: ***for*** $d \in \mathbf{N}_G(i)$ ***do***

If $|M_d| < n$, pad M_d with fake, but parsable, messages until it is length n (messages that start with the party number being 0).
for $d \in \mathbf{N}_G(i)$ **do**
  Let $k = 0$, $E_d = \varnothing$.
  for $m \in M_d$ **do**
    if $\mathtt{sim} = 0$ **then**
      Add $\mathsf{AE.Enc}_{pk}(m; \mathsf{prf}_{rk}(M_d, k, r))$ to E_d. // Used in protocol
    else
      Add $\mathsf{AE.Enc}_{pk}(m; r)$ to E_d. // Used in simulator
return $\{(E_d, d) : d \in \mathbf{N}_G(i)\}$

2.16 Proof of Theorem 9

Proof. **Simulator.** The simulator has to simulate the view of all corrupted parties. It knows the neighborhood of corrupted parties, its delay distributions and its clock rates. The view of the corrupted parties in the real world, consist of messages received from the network functionality $\mathcal{F}_{\text{NET}}$, and messages received from the hardware functionality $\mathcal{F}_{\text{HW}}$.

Consider a corrupted component C and the subgraph G_C, which contains C and the honest parties in the immediate neighborhood of C (but not the edges between honest parties). Observe that $\mathcal{S}_{HW}$ has complete knowledge of the topology of G_C.

To simulate the preprocessing phase, we do as follows: Each honest party in G_C starts with a state where its only neighbors are in G_C, and the simulated hardware box answers the queries exactly the same way as the real hardware box $\mathcal{F}_{\text{HW}}$. As a result, at the end of the preprocessing phase, all parties in G_C have an encrypted state containing the graph G_C.

In the computation phase, the simulator generates all local delays upfront. It then computes, for each corrupted party P_j in G_C, the number of traversals it initiates (recall that the party initiates a traversal every round, until the first message is received). For each of these traversals, it computes the last honest party P_i in G_C, before the traversal enters G_C (by executing the algorithm EulerianCycle on G_C), the next (corrupted) party P_k on the traversal, and samples and records the time at which the message arrives to P_i (corresponding to four times the total rounded delay minus the sum of rounded delays of the last fragment of corrupted parties in G_C). Then, the simulator checks at every round whether he has to send to a neighbor P_k a message containing the output of any corrupted party $P_j \in G_C$. It then sends the corresponding encryptions containing outputs, and appends encryptions so that every round, P_i sends n encryptions to each (corrupted) neighbor in G_C. The simulated hardware messages are as in the real protocol, except that the vector of inputs is 0, and, once the traversal is completed, it gives directly the output (ignoring the inputs). Moreover, it generates truly random values instead of using a PRF.

[17] The additional input $\mathtt{sim} \in \{0, 1\}$ will be used by the simulator and can be ignored at this point.

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

1. $\mathcal{S}_{HW}$ corrupts $\mathcal{Z}$.
2. $\mathcal{S}_{HW}$ sends inputs for all parties in $\mathcal{Z}$ to $\mathcal{F}_{\text{BC}}$ and for each party $P_i \in \mathcal{Z}$ receives the output value v_i^{out}.
3. Let R be the round length. $\mathcal{S}_{HW}$ receives from $\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{\mathsf{sum}}}$ the distribution $D = \sum_e \lceil D_e/R \rceil R$ and, for each $P_i \in \mathcal{Z}$, the neighborhood $\mathbf{N}_G(P_i)$ and its delay distributions.
4. $\mathcal{S}_{HW}$ generates an authenticated encryption key ek.
5. Now, $\mathcal{S}_{HW}$ has to simulate the view of all parties in $\mathcal{Z}$. The view of corrupted parties consist of messages received via the network $\mathcal{F}_{\text{NET}}$ and the queries to the hardware functionality $\mathcal{F}_{\text{HW}}$.

Network messages: The messages sent by corrupted parties can be easily generated by executing the protocol Hardware. To simulate the messages sent by an honest party to the corrupted neighbors, $\mathcal{S}_{HW}$ proceeds as follows.
First, it prepares a set `buffer`, containing all messages which will be sent by the honest parties to corrupted neighbors throughout the simulation (recall the variable `buffer` in $\mathcal{F}_{\text{NET}}$).
$\mathcal{S}_{HW}$ sets $\texttt{buffer} = \varnothing$. We simulate the messages per corrupted connected component. Let $G_C = (V_C, E_C) \subset G$ be a corrupted connected component including the honest parties in its immediate neighborhood. Then, $\mathcal{S}_{HW}$ does the following:

Preprocessing: Let P_j be a corrupted party in the component, who has an honest neighbor P_i.

1: Let $t_{\texttt{prep}}$ be the time when the preprocessing finishing signal is received by P_i.
2: Choose a random value $\texttt{id}_i$ and compute the initial vector `ID` as a vector of n $\perp$'s with $\texttt{id}_i$ in the i-th position. Let $s_0 = (i, \texttt{ID}, \mathbf{N}_{G_C}(P_i))$.
3: **for** $\tau \in \{0, R, 2R, \ldots, \lfloor t_{\texttt{prep}}/R \rfloor R\}$ **do**
4: Sample the delay d_{ij} from the distribution $D_{(i,j)}$ and record the tuple $(\tau + d_{ij}, P_i, P_j, \mathsf{AE.Enc}_{ek}(s_0))$ in `buffer`.

Computation:

1: Let $t_{\texttt{start}}$ be the start time of the computation (rounded to the next multiple of R), and for each party $P_i \in G_C$, let $t^i_{\texttt{end}}$ be the signal to terminate the execution of the computation phase.
2: For each honest party $P_i \in G_C$ and corrupted neighbor $P_j \in G_C$, set $S_{ij} = \varnothing$.
// Local delays generated upfront.
3: For each party $P_i \in G_C$ and neighbor $P_j \in G_C$, let $L_{(i,j)}$ be a list containing $n \cdot \left(\lfloor \frac{t^i_{\texttt{end}} - t_{\texttt{start}}}{R} \rfloor + 1 \right)$ samples from $D_{(i,j)}$. The messages sent by corrupted parties use these delays.
4: For each P_j, let $t^j_{\texttt{stop}} := \min\{t \in L_{(i,j)} : P_i \in \mathbf{N}_G(P_j)\}$ the time at which P_j obtains the first message from any neighbor (stops initiating Eulerian Cycles).

5: Compute the number of Eulerian Cycles initiated from each corrupted $P_j \in G_C$ as $N_j := \lfloor \frac{t^j_{\texttt{stop}}}{R} \rfloor$.

6: For each corrupted $P_j \in G_C$, do as follows: Run $\mathcal{E}^j = \text{EulerianCycle}(P_j, G_C)$. Let $\mathcal{E}^j_C$ be the starting path of the eulerian cycle from P_j until the first honest party P_i. Let P_k be the last corrupted party in $\mathcal{E}^j_C$. Let P'_j be the first party after P_j in the path.
For each $v \in [N_j]$, for each edge $e \in \mathcal{E}^j_C$, let d_e be the next unused delay. Then, sample $t^v_{\texttt{out}}$ from the distribution $4D - \sum_{e \in \mathcal{E}^j_C} \lceil d_e/R \rceil R$. Add $(P_j, t^v_{\texttt{out}})$ to S_{ik}.

7: Let P_i be an honest party, and let P_k be a corrupted neighbor. We simulate all messages sent by P_i to P_k. Let $S = \varnothing$.

8: // Message sent at Round 0

9: For each $w \in [n]$, record the tuple $(t_{\texttt{start}} + L_{(i,k)}[w], P_i, P_k, \mathsf{AE.Enc}_{ek}(\perp))$ in $\texttt{buffer}$, where $\perp$ is a parsable fake state.

10: // Messages sent at Round r

11: Let $\mathsf{r} = 1$.

12: **while** $t_{\texttt{start}} + \mathsf{r}R < t^i_{\texttt{end}}$ **do**

13: Let $S = \varnothing$ and let $\texttt{x}$ be a vector of length n initialized to 0.

14: If there exists $(P_j, t_{\texttt{out}}) \in S_{ik}$ such that $(\mathsf{r}-1)R \leq t_{\texttt{out}} < \mathsf{r}R$, add $\mathsf{AE.Enc}_{ek}((j, \texttt{ID}, G_C, \texttt{Path}, \texttt{x}, f_j))$ to S, where $\texttt{ID}$ is the vector of IDs that parties in G_C generated during the preprocessing, and $\texttt{Path} = \mathcal{E}^j \setminus \mathcal{E}^j_C$.

15: **while** $|S| < n$ **do**

16: Add an encryption of a parsable fake state $\mathsf{AE.Enc}_{ek}(\perp; z)$ to S.

17: Let $S = \{c_1, \ldots, c_n\}$. For each $v \in [n]$, record the tuple $(t_{\texttt{start}} + \mathsf{r}R + L_{(i,k)}[\mathsf{r}n + v], P_i, P_k, c_v)$ in $\texttt{buffer}$.

18: $\mathsf{r} = \mathsf{r} + 1$.

$\mathcal{S}_{HW}$ simulates the messages received by corrupted parties from $\mathcal{F}_{\text{NET}}$ as follows. On every input $(\text{FETCHMESSAGES}, j)$ from a corrupted P_j, it gets the current time τ from $\mathcal{F}_{\text{CLOCK}}$. Then, for each message tuple (T, P_i, P_j, c) from $\texttt{buffer}$ where $T \leq \tau$, it removes the tuple from $\texttt{buffer}$ and outputs (i, c) to P_j.

Hardware messages: $\mathcal{S}_{HW}$ has to simulate the replies to queries of corrupted parties to the hardware box functionality $\mathcal{F}_{\text{HW}}$. $\mathcal{S}_{HW}$ simulates the queries of the preprocessing phase doing exactly the same as the functionality $\mathcal{F}_{\text{HW}}$.
In the computation phase, on input $x = (i, \texttt{id}, c, E, x_i, f_i, z)$, $\mathcal{S}_{HW}$ simulates the query as follows: It executes the code of $\mathcal{F}_{\text{HW}}$, except that in Step 17, instead of evaluating f_i, it sets $\texttt{val} = v^{out}_j$, and in Step 20, it outputs ($\texttt{val}$, EncryptAndFormatOutput$(i, G, z, S, 1)$) (i.e., it sets $\texttt{sim} = 1$).

We now show that the execution with $\mathcal{S}_{HW}$ is indistinguishable from the real execution. For that, we present a sequence of hybrids. In the following, we only consider the messages sent by an honest P_i to its corrupted neighbor P_k (messages between corrupted neighbors are trivial to simulate).

Hybrid 1. $\mathcal{S}^1_{HW}$ simulates the real world exactly. That is, $\mathcal{S}^1_{HW}$ has information on the entire communication graph, all edge delays and all clock rates. It simulates the messages exactly.

Hybrid 2. $\mathcal{S}^2_{HW}$ generates fresh random values instead of using the outputs from the prf.

Hybrid 3. $\mathcal{S}^3_{HW}$ is exactly as $\mathcal{S}^2_{HW}$, except the way the output value is generated upon querying the hardware box. It returns the output v_j^{out} of the corresponding party (ignoring the input vector), instead of evaluating its function $f_j(\mathbf{x})$ on the input vector.

Hybrid 4. $\mathcal{S}^4_{HW}$, generates the local delays of the messages during the computation phase upfront, but still generates all messages as in $\mathcal{S}^3_{HW}$. That is, instead of sampling the delays when the message is going to be sent, it calculates the number of delays that are needed for the entire computation, and generates all samples upfront.

Hybrid 5. $\mathcal{S}^5_{HW}$, generates all messages in the computation phase by sending artificial ciphertexts containing either the output or parsable fake encryptions. More concretely, instead of following the path according to the message received $(j, \mathtt{ID}, G, \mathtt{Path}, \mathtt{x}, f_j)$ from the previous neighbor, it generates a new message $(j, \mathtt{ID}, G_C, \mathtt{Path}_j, \mathtt{x}_j, f_j)$ containing the graph G_C, the ID's in G_C, and the fake path on G_C started from P_j, but with the same ending. The input vector $\mathtt{x}_j$ is the 0 vector.

Hybrid 6. $\mathcal{S}^6_{HW}$, generates all messages in the computation phase at the correct times at follows: for each corrupted party P_j that has an Eulerian Cycle $\mathcal{E}^j$ where the first honest party is P_i and the previous party is P_k, it computes the time $t^j_{\mathtt{stop}}$ at which P_j received the first message, the number of Eulerian Cycles N_j initiated by P_j, and a sample delay for the delay it takes for each initiated cycle message to arrive to P_i.

Hybrid 7. $\mathcal{S}^5_{HW}$: P_i starts the preprocessing phase with the state where its only neighbors are the corrupted neighbors in G_C. This Hybrid corresponds to $\mathcal{S}_{HW}$.

- Hybrids 1 and 2 are indistinguishable by the security of the prf.
- For Hybrids 2 and 3 to be indistinguishable, we need that $f_j(\mathbf{x}) = v_j^{out}$, where v_j^{out} is received from the ideal functionality, and $\mathbf{x}$ and f_j are the values contained in the message. Hence, we have to show that the message contains the actual function and inputs (and not modified values from the adversary). To see why this holds, observe that any message that enters the corrupted component and can be decrypted has been processed by at least one honest party during the *second* cycle traversal. This means that this message was actually sent to this party. Now these sent messages are always correct (recall the adversary is passive) and the box never changes them. Moreover, the adversary cannot produce any forged or modified messages due to the security of the authenticated encryption. Hence, the value f_j is correct and $\mathbf{x}$ contains inputs (again, correctly) provided by parties during the first traversal.

- Hybrids 3 and 4 are trivially identical.
- The only difference between the Hybrids 4 and 5 is in the content of the encrypted messages generated by the simulator. We argue that the output of the simulated hardware box is indistinguishable for these messages. Observe that the outputs of the hardware box can be either encryptions, or the output. Both are trivially indistinguishable, as long as the box outputs an encryption in Hybrid 4 if and only if it outputs an encryption in Hybrid 5. This is the case, because (1) the graph G in any message from an honest party is correct (by an argument analogous to the one we used to argue that the function f_j is correct), and (2) the part of `Path` remaining to traverse when the Eulerian cycle is computed on j and G is the same as the part of $\texttt{Path}_j$ remaining to traverse when it is computed on j and G_C.
- Hybrids 5 and 6 are trivially identical, since the preprocessing messages are always encryptions under the secret key of $\mathcal{F}_{\text{HW}}$.
- Hybrids 6 and 7 are indistinguishable by semantic security.

□ □

Chapter 3

Property Preserving Hashing

Blurb about PPH.

3.1 Overview

The problem of property-preserving hashing*, namely how to compress a large input in a way that preserves a class of its properties, is an important one in the modern age of massive data. In particular, the idea of property-preserving hashing underlies sketching [109, 106, 7, 49, 39], compressed sensing [43], locality-sensitive hashing [80], and in a broad sense, much of machine learning.*

As two concrete examples in theoretical computer science, consider universal hash functions [36] which can be used to test the equality of data points, and locality-sensitive hash functions [80, 79] which can be used to test the ℓ_p-distance between vectors. In both cases, we trade off accuracy in exchange for compression. For example, in the use of universal hash functions to test for equality of data points, one stores the hash $h(x)$ of a point x together with the description of the hash function h. Later, upon obtaining a point y, one computes $h(y)$ and checks if $h(y) = h(x)$. The pigeonhole principle tells us that mistakes are inevitable; all one can guarantee is that they happen with an acceptably small probability. More precisely, universal hash functions tell us that

$$\forall x \neq y \in D, \Pr[h \leftarrow \mathcal{H} : h(x) \neq h(y)] \geq 1 - \epsilon$$

for some small ϵ. A cryptographer's way of looking at such a statement is that it asks the adversary to pick x and y first; and evaluates her success w.r.t. a hash function chosen randomly from the family $\mathcal{H}$. In particular, the adversary has no information about the hash function when she comes up with the (potentially) offending inputs x and y. Locality-sensitive hash functions have a similar flavor of correctness guarantee.

The starting point of this work is that this definition of correctness is too weak in the face of adversaries with access to the hash function (either the description of the function itself or perhaps simply oracle access to its evaluation). Indeed, in the context of equality testing, we have by now developed several notions of robustness against such adversaries, in the form of pseudorandom functions (PRF) [65], universal one-way

hash functions (UOWHF) [111] and collision-resistant hash functions (CRHF). Our goal in this work is to expand the reach of these notions beyond testing equality; that is, our aim is to do unto property-preserving hashing what CRHFs did to universal hashing.

Several works have observed the deficiency of the universal hash-type definition in adversarial settings, including a wide range of recent attacks within machine learning in adversarial environments (e.g., [102, 84, 125, 115, 85]). Such findings motivate a rigorous approach to combatting adversarial behavior in these settings, a direction in which significantly less progress has been made. Mironov, Naor and Segev [105] showed interactive protocols *for sketching in such an adversarial environment; in contrast, we focus on non-interactive hash functions. Hardt and Woodruff [68] showed negative results which say that linear functions cannot be robust (even against computationally bounded adversaries) for certain natural ℓ_p distance properties; our work will use non-linearity and computational assumptions to overcome the [68] attack. Finally, Naor and Yogev [110] study adversarial Bloom filters which compress a set in a way that supports checking set membership; we will use their lower bound techniques in Section 3.5.*

Motivating Robustness: Facial Recognition. *In the context of facial recognition, authorities A and B store the captured images x of suspects. At various points in time, say authority A wishes to look up B's database for a suspect with face x. A can do so by comparing $h(x)$ with $h(y)$ for all y in B's database.*

This application scenario motivated prior notions of fuzzy extractors and secure sketching. As with secure sketches and fuzzy extractors, a locality-sensitive property-preserving hash guarantees that close inputs (facial images) remain close when hashed [53]; this ensures that small changes in ones appearance do not affect whether or not that person is authenticated. However, neither fuzzy extractors nor secure sketching guarantees that far *inputs remain far when hashed. Consider an adversarial setting, not where a person wishes to evade detection, but where she wishes to be mistaken for someone else. Her face x' will undoubtably be different (far) from her target x, but there is nothing preventing her from slightly altering her face and passing as a completely different person when using a system with such a one-sided guarantee. This is where our notion of robustness comes in (as well as the need for cryptography): not only will adversarially chosen close x and x' map to close $h(x)$ and $h(x')$, but if adversarially chosen x and x' are* far*, they will be mapped to far outputs, unless the adversary is able to break a cryptographic assumption.*

Comparison to Secure Sketches and Fuzzy Extractors. *It is worth explicitly comparing fuzzy extractors and secure sketching to this primitive [53], as they aim to achieve similar goals. Both of these seek to preserve the privacy of their inputs. Secure sketches generate random-looking sketches that hide information about the original input so that the original input can be reconstructed when given something close to it. Fuzzy extractors generate uniform-looking keys based off of fuzzy (biometric) data also using entropy: as long as the input has enough entropy, so will the output. As stated above, both guarantee that if inputs are close, they will 'sketch' or 'extract' to the same object. Now, the entropy of the sketch or key guarantees that randomly*

generated far inputs will not collide, but there are no guarantees about adversarially generated far inputs. To use the example above, it could be that once an adversary sees a sketch or representation, she can generate two far inputs that will reconstruct to the correct input.

Robust Property-Preserving Hash Functions. *We put forth several notions of* robustness *for property-preserving hash (PPH) functions which capture adversaries with increasing power and access to the hash function. We then ask which properties admit robust property-preserving hash functions, and show positive and negative results.*

- *On the negative side, using a connection to communication complexity, we show that most properties and even simple ones such as set disjointness, inner product and greater-than do not admit non-trivial property-preserving hash functions.*
- *On the positive side, we provide two constructions of robust property-preserving hash functions (satisfying the strongest of our notions). The first is based on the standard cryptographic assumption of collision-resistant hash functions, and the second achieves more aggressive parameters under a new assumption related to the hardness of syndrome decoding on low density parity-check (LDPC) codes.*
- *Finally, we show that for essentially any non-trivial predicate (which we call collision-sensitive), achieving even a mild form of robustness requires cryptographic assumptions.*

We proceed to describe our contributions in more detail.

3.1.1 Our Results and Techniques

We explore two notions of properties. The first is that of property classes $\mathcal{P} = \{P : D \to \{0,1\}\}$*, sets of single-input predicates. This notion is the most general, and is the one in which we prove lower bounds. The second is that of two-input properties* $P : D \times D \to \{0,1\}$*, which compares two inputs. This second notion is more similar to standard notions of universal hashing and collision-resistance, stronger than the first, and where we get our constructions. We note that a two-input predicate has an analogous predicate-class* $\mathcal{P} = \{P_x\}_{x \in D}$*, where* $P_{x_1}(x_2) = P(x_1, x_2)$*.*

The notion of a property can be generalized in many ways, allowing for promise properties which output 0*,* 1 *or* $\circledast$ *(a don't care symbol), and allowing for more than* 2 *inputs. The simplest notion of correctness for property-preserving hash functions requires that, analogously to universal hash functions,*

$$\forall x, y \in D \Pr[h \leftarrow \mathcal{H} : \mathcal{H}.\mathsf{Eval}(h, h(x), h(y)) \neq P(x, y)] = \mathrm{negl}(\kappa)$$

or for single-input predicate-classes

$$\forall x \in D \text{ and } P \in \mathcal{P} \Pr[h \leftarrow \mathcal{H} : \mathcal{H}.\mathsf{Eval}(h, h(x), P) \neq P(x)] = \mathrm{negl}(\kappa)$$

where κ *is a security parameter.*

For the sake of simplicity in our overview, we will focus on two-input predicates.

Defining Robust Property-Preserving Hashing.

We define several notions of robustness *for PPH, each one stronger than the last. Here, we describe the strongest of all, called* direct-access PPH.

In a direct-access PPH, the (polynomial-time) adversary is given the hash function and is asked to find a pair of bad inputs, namely $x, y \in D$ such that $\mathcal{H}.\mathsf{Eval}(h, h(x), h(y)) \neq P(x, y)$. That is, we require that

$$\forall \text{ p.p.t. } \mathcal{A}, \Pr[h \leftarrow \mathcal{H}; (x, y) \leftarrow \mathcal{A}(h) : \mathcal{H}.\mathsf{Eval}(h, h(x), h(y)) \neq P(x, y)] = \mathrm{negl}(\kappa).$$

The direct-access definition is the analog of collision-resistant hashing for general properties.

Our other definitions vary by how much access the adversary is given to the hash function, and are motivated by different application scenarios. From the strong to weak, these include double-oracle PPH where the adversary is given access to a hash oracle and a hash evaluation oracle, and evaluation-oracle PPH where the adversary is given only a combined oracle. Definitions similar to double-oracle PPH have been proposed in the context of adversarial bloom filters [110], and ones similar to evaluation-oracle PPH have been proposed in the context of showing attacks against property-preserving hash functions [68]. For more details, we refer the reader to Section **??**.

Connections to Communication Complexity and Negative Results. *Property-preserving hash functions for a property P, even without robustness, imply communication-efficient protocols for P in several models. For example, any PPH for P implies a protocol for P in the simultaneous messages model of Babai, Gal, Kimmel and Lokam [12] wherein Alice and Bob share a common random string h, and hold inputs x and y respectively. Their goal is to send a single message to Charlie who should be able to compute $P(x, y)$ except with small error. Similarly, another formalization of PPH that we present, called PPH for single-input predicate classes (see Section* **??***) implies efficient protocols in the one-way communication model [134].*

We use known lower bounds in these communication models to rule out PPHs for several interesting predicates (even without robustness). There are two major differences between the PPH setting and the communication setting, however: (a) in the PPH setting, we demand an error that is negligible (in a security parameter); and (b) we are happy with protocols that communicate $n - 1$ bits (or the equivalent bound in the case of promise properties) whereas the communication lower bounds typically come in the form of $\Omega(n)$ bits. In other words, the communication lower bounds as-is *do not rule out PPH.*

At first thought, one might be tempted to think that the negligible-error setting is the same as the deterministic setting where there are typically lower bounds of n (and not just $\Omega(n)$); however, this is not the case. For example, the equality function which has a negligible error public-coin simultaneous messages protocol (simply using universal hashing) with communication complexity $CC = O(\kappa)$ and deterministic protocols require $CC \geq n$. Thus, deterministic lower bounds do not (indeed, cannot)

do the job, and we must better analyze the randomized lower bounds. Our refined analysis shows the following lower bounds:

- *PPH for the Gap-Hamming (promise) predicate with a gap of $\sqrt{n}/2$ is impossible by refining the analysis of a proof by Jayram, Kumar and Sivakumar [81]. The Gap-Hamming predicate takes two vectors in $\{0,1\}^n$ as input, outputs 1 if the vectors are very far, 0 if they are very close, and we do not care what it outputs for inputs in the middle.*

- *We provide a framework for proving PPHs are impossible for some total predicates, characterizing these classes as* reconstructing. *A predicate-class is* reconstructing *if, when only given oracle access to the predicates of a certain value x, we can efficiently determine x with all but negligible probability.*[1] *With this framework, we show that PPH for the Greater-Than (GT) function is impossible. It was known that GT required $\Omega(n)$ bits (for constant error) [116], but we show a lower bound of exactly n if we want negligible error. Index and Exact-Hamming are also reconstructing predicates.*

- *We also obtain a lower bound for a variant of GT: the (promise) Gap-k GT predicate which on inputs $x, y \in [N = 2^n]$, outputs 1 if $x - y > k$, 0 if $y - x > k$, and we do not care what it outputs for inputs in between. Here, exactly $n - \log(k) - 1$ bits are required for a perfect PPH. This is tight: we show that with fewer bits, one cannot even have a non-robust PPH, whereas there is a perfect robust PPH that compresses to $n - \log(k) - 1$ bits.*

New Constructions. *Our positive results are two constructions of a direct-access PPH for gap-Hamming for n-length vectors for large gaps of the form $\sim O(n/\log n)$ (as opposed to an $O(\sqrt{n})$-gap for which we have a lower bound). Let us recall the setting: the gap Hamming predicate P_{ham}, parameterized by n, d and ϵ, takes as input two n-bit vectors x and y, and outputs 1 if the Hamming distance between x and y is greater than $d(1+\epsilon)$, 0 if it is smaller than $d(1-\epsilon)$ and a don't care symbol $\circledast$ otherwise. To construct a direct-access PPH for this (promise) predicate, one has to construct a compressing family of functions $\mathcal{H}$ such that*

$$\forall \text{ p.p.t. } \mathcal{A}, \Pr[h \leftarrow \mathcal{H}; (x,y) \leftarrow \mathcal{A}(h) : P_{\mathsf{ham}}(x,y) \neq \circledast \\ \wedge \mathcal{H}.\mathsf{Eval}(h, h(x), h(y)) \neq P_{\mathsf{ham}}(x,y)] = \mathrm{negl}(\kappa). \tag{3.1}$$

Our two constructions offer different benefits. The first provides a clean general approach, and relies on the standard cryptographic assumption of collision-resistant hash functions. The second builds atop an existing one-way communication protocol, supports a smaller gap and better efficiency, and ultimately relies on a (new) variant of the syndrome decoding assumption on low-density parity check codes.

Construction 1. *The core idea of the first construction is to reduce the goal of robust Hamming PPH to the simpler one of robust equality testing; or, in a word,*

[1] In the single-predicate language of above, the predicate class corresponds to $\mathcal{P} = \{P(x, \cdot)\}$.

"subsampling." The intuition is to notice that if $\mathbf{x}_1 \in \{0,1\}^n$ *and* $\mathbf{x}_2 \in \{0,1\}^n$ *are* close*, then* most small enough subsets *of indices of* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *will match identically. On the other hand, if* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *are* far*, then most* large enough subsets *of indices will differ.*

The hash function construction will thus fix a collection of sets $\mathcal{S} = \{S_1, \ldots, S_k\}$*, where each* $S_i \subseteq [n]$ *is a subset of appropriately chosen size* s*. The desired structure can be achieved by defining the subsets* S_i *as the neighbor sets of a bipartite expander. On input* $\mathbf{x} \in \{0,1\}^n$*, the hash function will consider the vector* $\mathbf{y} = (\mathbf{x}|_{S_1}, \ldots, \mathbf{x}|_{S_k})$ *where* $\mathbf{x}|_S$ *denotes the substring of* $\mathbf{x}$ *indexed by the set* S*. The observation above tells us that if* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *are close (resp. far), then so are* $\mathbf{y}_1$ *and* $\mathbf{y}_2$*.*

Up to now, it is not clear that progress has been made: indeed, the vector $\mathbf{y}$ *is not compressing (in which case, why not stick with* $\mathbf{x}_1, \mathbf{x}_2$ *themselves?). However,* $\mathbf{y}_1, \mathbf{y}_2$ *satisfy the desired Hamming distance properties with fewer symbols over a* large alphabet*,* $\{0,1\}^s$*. As a final step, we can then leverage (standard) collision-resistant hash functions (CRHF) to compress these symbols. Namely, the final output of our hash function* $h(\mathbf{x})$ *will be the vector* $(g(\mathbf{x}|_{S_1}), \ldots, g(\mathbf{x}|_{S_k}))$*, where each substring of* $\mathbf{x}$ *is individually compressed by a CRHF* g*.*

The analysis of the combined hash construction then follows cleanly via two steps. The (computational) collision-resistence property of g *guarantees that any efficiently found pair of inputs* $\mathbf{x}_1, \mathbf{x}_2$ *will satisfy that their hash outputs*

$$h(\mathbf{x}_1) = (g(\mathbf{x}_1|_{S_1}), \ldots, g(\mathbf{x}_1|_{S_k})) \quad \textit{and} \quad h(\mathbf{x}_2) = (g(\mathbf{x}_2|_{S_1}), \ldots, g(\mathbf{x}_2|_{S_k}))$$

are close if and only if it holds that

$$(\mathbf{x}_1|_{S_1}, \ldots, \mathbf{x}_1|_{S_k}) \quad \textit{and} \quad (\mathbf{x}_2|_{S_1}, \ldots, \mathbf{x}_2|_{S_k})$$

are close as well; that is, $\mathbf{x}_1|_{S_i} = \mathbf{x}_2|_{S_i}$ *for most* S_i*. (Anything to the contrary would imply finding a collision in* g*.) Then, the combinatorial properties of the chosen index subsets* S_i *ensures (unconditionally) that any such inputs* $\mathbf{x}_1, \mathbf{x}_2$ *must themselves be close. The remainder of the work is to specify appropriate parameter regimes for which the CRHF can be used and the necessary bipartite expander graphs exist.*

Construction 2. *The starting point for our second construction is a simple non-robust hash function derived from a one-way communication protocol for gap-Hamming due to Kushilevitz, Ostrovsky, and Rabani [91]. In a nutshell, the hash function is parameterized by a random sparse* $m \times n$ *matrix* A *with* 1*'s in* $1/d$ *of its entries and* 0*'s elsewhere; multiplying this matrix by a vector* $\mathbf{z}$ *"captures" information about the Hamming weight of* $\mathbf{z}$*. However, this can be seen to be trivially* not robust *when the hash function is given to the adversary. The adversary simply performs Gaussian elimination, discovering a "random collision"* (x, y) *in the function, where, with high probability* $x \oplus y$ *will have large Hamming weight. This already breaks equation (3.1).*

The situation is somewhat worse. Even in a very weak, oracle sense, corresponding to our evaluation-oracle-robustness definition, a result of Hardt and Woodruff [68] shows that there are no linear functions h *that are robust for the gap-*ℓ_2 *predicate. While their result does not carry over* as-is *to the setting of* ℓ_0 *(Hamming), we conjecture it does, leaving us with two options: (a) make the domain sparse: both the*

Gaussian elimination attack and the Hardt-Woodruff attack use the fact that Gaussian elimination is easy on the domain of the hash function; however making the domain sparse (say, the set of all strings of weight at most βn for some constant $\beta < 1$) already rules it out; and (b) make the hash function non-linear: again, both attacks crucially exploit linearity. We will pursue both options, and as we will see, they are related.

But before we get there, let us ask whether we even need computational assumptions to get such a PPH. Can there be information-theoretic constructions? The first observation is that by a packing argument, if the output domain of the hash function has size less than $2^{n-n\cdot H(\frac{d(1+\epsilon)}{n})} \approx 2^{n-d\log n(1+\epsilon)}$ (for small d), there are bound to be "collisions", namely, two far points (at distance more than $d(1+\epsilon)$) that hash to the same point. So, you really cannot compress much information-theoretically, especially as d becomes smaller. A similar bound holds when restricting the domain to strings of Hamming weight at most βn for constant $\beta < 1$.

With that bit of information, let us proceed to describe in a very high level our construction and the computational assumption. Our construction follows the line of thinking of Applebaum, Haramaty, Ishai, Kushilevitz and Vaikuntanathan [10] where they used the hardness of syndrome decoding problems to construct collision-resistant hash functions. Indeed, in a single sentence, our observation is that their collision-resistant hash functions are locality-sensitive *by virtue of being* input-local, *and thus give us a robust gap-Hamming PPH (albeit under a different assumption).*

In slightly more detail, our first step is to simply take the construction of Kushilevitz, Ostrovsky, and Rabani [91], and restrict the domain of the function. We show that finding two close points that get mapped to far points under the hash function is simply impossible (for our setting of parameters). On the other hand, there exist two far points that get mapped to close points under the hash functions (in fact, they even collide). Thus, showing that it is hard to find such points requires a computational assumption.

In a nutshell, our assumption says that given a random matrix $\mathbf{A}$ where each entry is chosen from the Bernoulli distribution with $\mathsf{Ber}(1/d)$ with parameter $1/d$, it is hard to find a large Hamming weight vector $\mathbf{x}$ where $\mathbf{Ax} \pmod 2$ has small Hamming weight. Of course, "large" and "small" here have to be parameterized correctly (see Section 3.4.9 for more details), however we observe that this is a generalization of the syndrome decoding assumption for low-density parity check (LDPC) codes, made by [10].

In our second step, we remove the sparsity requirement on the input domain of the predicate. We show a sparsification transformation which takes arbitrary n-bit vectors and outputs $n' > n$-bit sparse *vectors such that (a) the transformation is injective, and (b) the expansion introduced here does not cancel out the effect of compression achieved by the linear transformation $\mathbf{x} \to \mathbf{Ax}$. This requires careful tuning of parameters for which we refer the reader to Section 3.4.9.*

The Necessity of Cryptographic Assumptions. *The goal of robust PPH is to compress beyond the information theoretic limits, to a regime where incorrect hash outputs exist but are hard to find. If robustness is required even when the hash function*

is given, this inherently necessitates cryptographic hardness assumptions. A natural question is whether weaker forms of robustness (where the adversary sees only oracle access to the hash function) similarly require cryptographic assumptions, and what types of assumptions are required to build non-trivial PPHs of various kinds.

As a final contribution, we identify necessary assumptions for PPH for a kind of predicate we call collision sensitive*. In particular, PPH for any such predicate in the double-oracle model implies the existence of one-way functions, and in the direct-access model implies existence of collision-resistant hash functions. In a nutshell, collision-sensitive means that finding a collision in the predicate breaks the property-preserving nature of any hash. The proof uses and expands on techniques from the work of Naor and Yogev on adversarially robust Bloom Filters [110]. The basic idea is the same: without OWFs, we can invert arbitrary polynomially-computable functions with high probability in polynomial time, and using this we get a representation of the hash function/set, which can be used to find offending inputs.*

3.2 Preliminaries for PPH

TODO

3.3 Defining Property-Preserving Hash Functions

Our definition of property preserving hash functions (PPHs) comes in several flavors, depending on whether we support total or partial predicates; whether the predicates take a single input or multiple inputs; and depending on the information available to the adversary. We discuss each of these choices in turn.

Total vs. Partial Predicates. *We consider* total predicates *that assign a* 0 *or* 1 *output to each element in the domain, and* promise (or partial) predicates *that assign a* 0 *or* 1 *to a subset of the domain and a wildcard (don't-care) symbol* $\circledast$ *to the rest. More formally, a* total predicate P *on a domain* X *is a function* $P : X \to \{0, 1\}$*, well-defined as* 0 *or* 1 *for every input* $x \in X$*. A* promise predicate P *on a domain* X *is a function* $P : X \to \{0, 1, \circledast\}$*. Promise predicates can be used to describe scenarios (such as gap problems) where we only care about providing an exact answer on a subset of the domain.*

Our definitions below will deal with the more general case of promise predicates, but we will discuss the distinction between the two notions when warranted.

Single-Input vs Multi-Input Predicates. *In the case of single-input predicates, we consider a class of properties* $\mathcal{P}$ *and hash a single input* x *into* $h(x)$ *in a way that given* $h(x)$*, one can compute* $P(x)$ *for any* $P \in \mathcal{P}$*. Here,* h *is a compressing function. In the multi-input setting, we think of a single fixed property* P *that acts on a tuple of inputs, and require that given* $h(x_1), h(x_2), \ldots, h(x_k)$*, one can compute* $P(x_1, x_2, \ldots, x_k)$*. The second syntax is more expressive than the first, and so we use the multi-input syntax for constructions and the single-input syntax for lower bounds*[2]*.*

[2]There is yet a third possibility, namely where there is a *fixed* predicate P that acts on a single

Before we proceed to discuss robustness, we provide a working definition for a property-preserving hash function for the single-input syntax. For the multi-input predicate definition and further discussion, see appendix 3.3.8.

Definition 9. *A (non-robust) η-compressing Property Preserving Hash (η-PPH) family $\mathcal{H} = \{h : X \to Y\}$ for a function η and a class of predicates $\mathcal{P}$ requires the following two efficiently computable algorithms:*

- $\mathcal{H}.\mathsf{Samp}(1^\kappa) \to h$ *is a randomized p.p.t. algorithm that samples a random hash function from $\mathcal{H}$ with security parameter κ.*
- $\mathcal{H}.\mathsf{Eval}(h, P, y)$ *is a deterministic polynomial-time algorithm that on input the hash function h, a predicate $P \in \mathcal{P}$ and $y \in Y$ (presumably $h(x)$ for some $x \in X$), outputs a single bit.*

Additionally, $\mathcal{H}$ must satisfy the following two properties:

- η-compressing, *namely,* $\log |Y| \leq \eta(\log |X|)$, *and*
- robust, *according to one of four definitions that we describe below, leading to four notions of PPH: definition 10 (non-robust PPH), 11 (evaluation-oracle-robust PPH or EO-PPH), 12 (double-oracle-robust PPH or DO-PPH), or 13 (direct-access robust PPH or DA-PPH). We will refer to the strongest form, namely direct-access robust PPH as simply robust PPH when the intent is clear from the context. See also figure 3-1 for a direct comparison between these.*

The Many Types of Robustness. *We will next describe four definitions of robustness for PPHs, starting from the weakest to the strongest. Each of these definitions, when plugged into the last bullet of Definition 9, gives rise to a different type of property-preserving hash function. In each of these definitions, we will describe an adversary whose goal is to produce an input and a predicate such that the hashed predicate evaluation disagrees with the truth. The difference between the definitions is in what an adversary has access to, summarized in figure 3-1.*

3.3.1 Non-Robust PPH

We will start by defining the weakest notion of robustness which we call non-robust PPH. Here, the adversary has no information at all on the hash function h, and is required to produce a predicate P and a valid input x, namely where $P(x) \neq \circledast$, such that $\mathcal{H}.\mathsf{Eval}(h, P, x) \neq P(x)$ with noticeable probability. When $\mathcal{P}$ is the family of point functions (or equality functions), this coincides with the notion of 2-universal hash families [36][3].

input x, and we require that given $h(x)$, one can compute $P(x)$. This makes sense when the computational complexity of h is considerably less than that of P, say when P is the parity function and h is an AC^0 circuit, as in the work of Dubrov and Ishai [54]. We do not explore this third syntax further in this work.

[3]While 2-universal hashing corresponds with a two-input predicate testing equality, the single-input version ($\{P_{x_1}\}$ where $P_{x_1}(x_2) = (x_1 == x_2)$) is more general, and so it is what we focus on.

For security parameter λ, fixed predicate class $\mathcal{P}$, and h sampled from $\mathcal{H}.\mathsf{Samp}$

Non-Robust PPH	Adversary has no access to hash function or evaluation.
Evaluation-Oracle PPH	Access to the evaluation oracle $\mathcal{O}_h^{\mathsf{Eval}}(x, P) = \mathcal{H}.\mathsf{Eval}(h, P, h(x))$.
Double-Oracle PPH	Access to both $\mathcal{O}_h^{\mathsf{Eval}}$ (as above) and hash oracle $\mathcal{O}_h^{\mathsf{Hash}}(x) = h(x)$.
Robust PPH "Direct Access"	Direct access to the hash function, description of h.

Figure 3-1: A table comparing the adversary's access to the hash function within different robustness levels of PPHs.

Here and in the following, we use the notation $\Pr[A_1; \ldots; A_m : E]$ *to denote the probability that event* E *occurs following an experiment defined by executing the sequence* $A_1, \ldots, A_m$ *in order.*

Definition 10. *A family of PPH functions* $\mathcal{H} = \{h : X \to Y\}$ *for a class of predicates* $\mathcal{P}$ *is a family of* non-robust *PPH functions if for any* $P \in \mathcal{P}$ *and* $x \in X$ *such that for* $P(x) \neq \circledast$,

$$\Pr[h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\kappa) : \ \mathcal{H}.\mathsf{Eval}(h, P, h(x))) \neq P(x)] \leq \mathrm{negl}(\kappa).$$

3.3.2 Evaluation-Oracle Robust PPH

In this model, the adversary has slightly more power than in the non-robust setting. Namely, she can adaptively query an oracle that has $h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\kappa)$ *in its head, on inputs* $P \in \mathcal{P}$ *and* $x \in X$*, and obtain as output the hashed evaluation result* $\mathcal{H}.\mathsf{Eval}(h, P, h(x))$*. Let* $\mathcal{O}_h(x, P) = \mathcal{H}.\mathsf{Eval}(h, P, h(x))$*.*

Definition 11. *A family of PPH functions* $\mathcal{H} = \{h : X \to Y\}$ *for a class of predicates* $\mathcal{P}$ *is a family of* evaluation-oracle robust (EO-robust) *PPH functions if, for any PPT adversary* $\mathcal{A}$*,*

$$\begin{aligned}\Pr[h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\kappa);\ (x, P) \leftarrow \mathcal{A}^{\mathcal{O}_h}(1^\kappa) :\\ P(x) \neq \circledast \wedge \mathcal{H}.\mathsf{Eval}(h, P, h(x)) \neq P(x)] \leq \mathrm{negl}(\kappa).\end{aligned}$$

The reader might wonder if this definition is very weak, and may ask if it follows just from the definition of a non-robust PPH family. In fact, for total predicates*, we show that the two definitions are the same. At a high level, simply querying the evaluation oracle on (even adaptively chosen) inputs cannot reveal information about the hash function since with all but negligible probability, the answer from the oracle will be correct and thus simulatable without oracle access. The proof of the following lemma is in Appendix 3.3.6.*

Lemma 10. *Let* $\mathcal{P}$ *be a class of total predicates on* X*. A non-robust PPH* $\mathcal{H}$ *for* $\mathcal{P}$ *is also an Evaluation-Oracle robust PPH for* $\mathcal{P}$ *for the same domain* X *and same codomain* Y*.*

However, when dealing with promise *predicates, an EO-robustness adversary has the ability to make queries that do not satisfy the promise, and could get information about the hash function, perhaps even reverse-engineering the entire hash function itself. Indeed, Hardt and Woodruff [68] show that there are no EO-robust* linear *hash functions for a certain promise-ℓ_p distance property; whereas, non-robust linear hash functions for these properties follow from the work of Indyk [80, 79].*

3.3.3 Double-Oracle PPH

We continue our line of thought, giving the adversary more power. Namely, she has access to two oracles, both have a hash function $h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\kappa)$ in their head. The hash oracle $\mathcal{O}_h^{\mathsf{Hash}}$, parameterized by $h \in \mathcal{H}$, outputs $h(x)$ on input $x \in X$. The predicate evaluation oracle $\mathcal{O}_h^{\mathsf{Eval}}$, also parameterized by $h \in \mathcal{H}$, takes as input $P \in \mathcal{P}$ and $y \in Y$ and outputs $\mathcal{H}.\mathsf{Eval}(h, P, y)$. When $\mathcal{P}$ is the family of point functions (or equality functions), this coincides with the notion of psuedo-random functions.

Definition 12. *A family of PPH functions $\mathcal{H} = \{h : X \to Y\}$ for a class of predicates $\mathcal{P}$ is a family of* double-oracle-robust PPH (DO-PPH) *functions if, for any PPT adversary $\mathcal{A}$,*

$$\Pr[h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\kappa);\ (x, P) \leftarrow \mathcal{A}^{\mathcal{O}_h^{\mathsf{Hash}}, \mathcal{O}_h^{\mathsf{Eval}}}(1^\kappa) : P(x) \neq \circledast \wedge \mathcal{H}.\mathsf{Eval}(h, P, h(x)) \neq P(x)] \leq \mathrm{negl}(\kappa).$$

We show that any evaluation-oracle-robust PPH can be converted into a double-oracle-robust PPH at the cost of a computational assumption, namely, one-way functions. In a nutshell, the observation is that the output of the hash function can be encrypted using a symmetric key that is stored as part of the hash description, and the evaluation proceeds by first decrypting.

Lemma 11. *Let $\mathcal{P}$ be a class of (total or partial) predicates on X. Assume that one-way functions exist. Then, any EO-robust PPH for $\mathcal{P}$ can be converted into a DO-robust PPH for $\mathcal{P}$.*

See appendix 3.3.7 for the full proof.

3.3.4 Direct-Access Robust PPH

Finally, we define the strongest notion of robustness where the adversary is given the description of the hash function itself. When $\mathcal{P}$ is the family of point functions (or equality functions), this coincides with the notion of collision-resistant hash families.

Definition 13. *A family of PPH functions $\mathcal{H} = \{h : X \to Y\}$ for a class of predicates $\mathcal{P}$ is a family of* direct-access robust *PPH functions if, for any PPT adversary $\mathcal{A}$,*

$$\Pr[h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\kappa);\ (x, P) \leftarrow \mathcal{A}(h) : P(x) \neq \circledast \wedge \mathcal{H}.\mathsf{Eval}(h, P, h(x))) \neq P(x)] \leq \mathrm{negl}(\kappa).$$

We will henceforth focus on direct-access-robust *property-preserving hash functions and refer to them simply as robust PPHs.*

3.3.5 Proofs of Relationships between definitions

Here we will write the proofs for the lemmas described in section 2. First, lemma 10 states that for total predicates, non-robustness and evaluation-oracle PPHs are equivalent. Then, lemma 11 states that with OWFs, we can take a EO-robust PPH to get a DO-robust PPH, simply by pairing a hash function with an invertible PRF.

3.3.6 Proof of lemma 10: From a Non-Robust to Robust PPH for Total Predicates

Here is the proof that a non-robust PPH for a total predicate implies an Evaluation-Oracle PPH.

Lemma 12. *Let $\mathcal{P}$ be a class of total predicates on X. A non-robust PPH $\mathcal{H}$ for $\mathcal{P}$ is also an Evaluation-Oracle robust PPH for $\mathcal{P}$ for the same domain X and same codomain Y.*

Proof. Let $\mathcal{H} = \{h : X \to Y\}$ be a non-robust PPH for a class of total predicates $\mathcal{P}$ on X. Without any access to the hash function itself, any adversary (not even computationally bounded) has a negligible chance of coming up with an x and P that violate correctness because the adversary has no idea which h was sampled from $\mathcal{H}$. We will show that even given an Evaluation Oracle, $\mathcal{A}$ still cannot learn anything about which h was sampled, and so has the same advantage as blindly guessing.

Let $\mathcal{A}$ make at most T queries to $\mathcal{O}_h^{\mathsf{Eval}}$. Let $\mathcal{O}_{\mathcal{P}}$ just be the trivial predicate evaluation oracle, so $\mathcal{O}_{\mathcal{P}}(x, P) = P(x)$. We will now construct a series of t hybrids.

- Hybrid 0. $\mathcal{A}$ queries $\mathcal{O}_h^{\mathsf{Eval}}$.
- Hybrid t. For the first t queries, $\mathcal{A}$ gets answers from $\mathcal{O}_{\mathcal{P}}$. For the last $T - t$ queries, $\mathcal{A}$ gets answers from $\mathcal{O}_h^{\mathsf{Eval}}$.
- Hybrid T. $\mathcal{A}$ makes all T queries to $\mathcal{O}_{\mathcal{P}}$.

Note that $\mathcal{A}$'s first query to $\mathcal{O}_h^{\mathsf{Eval}}$ has a negligible chance of being answered incorrectly due to the correctness of the PPH (i.e. has a negligible chance of being distinguishable from $\mathcal{O}_{\mathcal{P}}$). The only way for $\mathcal{A}$ to distinguish hybrids $t-1$ and t is if query t was answered incorrectly. Since query t is $\mathcal{A}$'s first query to the $\mathcal{O}_h^{\mathsf{Eval}}$ in Hybrid t, $\mathcal{A}$ will be able to detect this difference with negligible probability in κ.

Since $T = \text{poly}(\kappa)$, a union bound yields that the maximum possible probability $\mathcal{A}$ can distinguish Hybrid 0 from Hybrid T is $\text{poly}(\kappa) \cdot \text{negl}(\kappa) = \text{negl}(\kappa)$.

So, in Hybrid T, $\mathcal{A}$ is making no queries to $\mathcal{O}_h^{\mathsf{Eval}}$. In fact, $\mathcal{A}$ can simulate every response from $\mathcal{O}_{\mathcal{P}}$ just by evaluating $P(x)$ on its own.

$$\Pr_{h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\kappa)}[\mathcal{A}^{\mathcal{O}_h^{\mathsf{Eval}}}(1^\kappa) \to (x, P) : P'(h(x)) \neq P(x)]$$

$$\Pr_{h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\kappa)}[\mathcal{A}(1^\kappa) \to (x, P) : P'(h(x)) \neq P(x)] + \text{negl}(\kappa) = \text{negl}(\kappa)$$

Therefore, $\mathcal{H}$ is secure in the Evaluation-Oracle model. □

Figure 3-2: Tranforming a PPH that is secure against adversaries that do not have access to the hash function and only oracle access to predicates to a PPH secure against adversaries with oracle access to the hash functions using CCA2-secure symmetric encryption.

Given $\mathcal{H}$ with algorithms $(\mathsf{Samp}, \mathsf{Transf})$ a no-function access, oracle-predicate PPH family and a CCA2-secure symmetric encryption scheme $(\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Decrypt})$, we can construct $\mathcal{H}^*$ with algorithms $(\mathsf{Samp}^*, \mathsf{Transf}^*)$ as follows.

$\mathsf{Samp}^*(1^\lambda)$:

1. $h \leftarrow \mathsf{Samp}(1^\lambda)$.
2. $(f_k, k) \xleftarrow{\$} \mathcal{F}$.
3. Output $h^* = (h, k)$ where $h^*(x) = f_k(h(x))$.

$\mathsf{Eval}^*(h^*, P, y^*)$

1. Parse $h^* = (h, k)$.
2. $y \leftarrow f_k^{-1}(y^*)$.
3. Output $\mathsf{Eval}(h, P, y)$.

3.3.7 Proof of lemma 11: Amplifying an EO-robust PPH to a DO-robust PPH

Here we will restate the lemma.

Lemma 13. *Let $\mathcal{P}$ be a class of (total or partial) predicates on X. Assume that one-way functions exist. Then, any EO-robust PPH for $\mathcal{P}$ can be converted into a DO-robust PPH for $\mathcal{P}$.*

Proof. First, let OWFs exist. Then, PRP's also exist. So, let $m = \eta n$, and $\mathcal{F} = \{f : \{0,1\}^n \to \{0,1\}^m\}$ be a family of *strong* PRP's where each f_k is efficiently invertible with the key k. The characterization of strong here means that f_k^{-1} is also a PRP. Figure 3-2 details how to take an EO-robust PPH $\mathcal{H}$ and get a DO-robust PPH $\mathcal{H}^*$. It is easy to see that $\mathcal{H}^*$ satisfies the efficiency properties of the sampling algorithm, and since $\mathcal{F}$ is a PRP, $\mathcal{H}^*$ is also η-compressing. We still need to prove that this is robust. We will do this with a series of hybrids. Let $\mathcal{A}$ be an adversary against $\mathcal{H}^*$. Let $\mathcal{B}$ run $\mathcal{A}$ as a subroutine and have access to $\mathcal{O}_h$. Let $T = \text{poly}(n)$ be the maximum number of queries $\mathcal{A}$ makes to $\mathcal{O}_{h^*}^{\mathsf{Hash}}$ and $\mathcal{O}_{h^*}^{\mathsf{Eval}}$ to break the correctness $\mathcal{H}^*$ with non-negligible probability.

- Hybrid 0. In this game, $\mathcal{A}$ makes all T hash and evaluation queries to $\mathcal{O}_{h^*}^{\mathsf{Hash}}$ and $\mathcal{O}_{h^*}^{\mathsf{Eval}}$ respectively. $\mathcal{B}$ outputs the (x, P) that $\mathcal{A}$ outputs at the end of its queries.

- Hybrid t. In this game, $\mathcal{A}$ makes the first $t-1$ queries to $\mathcal{O}_{h^*}^{\mathsf{Hash}}$ and $\mathcal{O}_{h^*}^{\mathsf{Eval}}$ appropriately. But, then $\mathcal{B}$ simulates every query from t to T as follows:

  - For every hash query x, if x had already been queried before, $\mathcal{B}$ just sends the same answer as given before by $\mathcal{O}_{h^*}^{\mathsf{Hash}}$. If x has not been queried before, $\mathcal{B}$ chooses a random element y in the image of h^* that has not been seen before. $\mathcal{B}$ saves the pair (x, y) in memory.
  - For every evaluation query y, if y is associated with some x as $h^*(x)$, then $\mathcal{B}$ queries $\mathcal{O}_h$ with the pair (x, P). $\mathcal{O}_h$ correctly returns $\mathcal{H}.\mathsf{Eval}(h, P, h(x)) = \mathcal{H}^*.\mathsf{Eval}(h^*, P, h^*(x))$. If y has not been associated with an x, $\mathcal{B}$ chooses a random element $x \in \{0,1\}$ that has not been queried/seen before, saves the pair (x, y). Then, $\mathcal{B}$ queries $\mathcal{O}_h(x, P)$.

  $\mathcal{B}$ outputs the (x, P) that $\mathcal{A}$ outputs at the end of its queries.

- Hybrid T. $\mathcal{B}$ simulates the answer to every single query $\mathcal{A}$ makes as follows just as above. $\mathcal{B}$ outputs the (x, P) that $\mathcal{A}$ outputs at the end of its queries.

If $\mathcal{B}$ has a non-negligible probability of outputting (x, P) breaking the correctness of $\mathcal{H}^*$ in Hybrid T, then either $\mathcal{A}$ has non-negligible probability of outputting some (x, P) in Hybrid 0, or there exists a $t \in [T]$ where $\mathcal{A}$ has a noticeable gap in winning Hybrid t versus winning Hybrid $t-1$. Therefore, we can create an adversary $\mathcal{A}^*$ that can distinguish, with non-negligible probability, between Hybrids t and $t-1$. Moreover, the query made at that point must be a query x or y that has not been asked about before (otherwise there is no difference between the Hybrids). If the query is a hash query, then this implies $\mathcal{A}^*$ can distinguish between the PRP $f_k(h(x))$ and a truly random permutation. This cannot happen because of the pseudorandomness of f_k. If the query is an evaluation query on a y that we have not yet seen, then we assume it was associated with a random x not yet queried; $\mathcal{A}^*$ is distinguishing between $f_k^{-1}(y)$ and random. Because f_k is a strong PRP, f_k^{-1} is also a PRP, and therefore, distinguishing $f_k^{-1}(y)$ from random should also be impossible for PPT adversaries.

So, since any PPT algorithm in finding (x, P) such that $\mathcal{H}.\mathsf{Eval}(h, h(x), P) \neq P(x)$, $\mathcal{B}$ must also have negligible advantage, and therefore $\mathcal{A}$ also has negligible advantage. □

3.3.8 Multi-Input vs Single-Input Predicates

We discuss the differences between definitions of property-preserving hashes with a family of single-input predicates versus a fixed multi-input predicate. For an example, consider the two-input equality predicate, namely $P(x_1, x_2) = 1$ if $x_1 = x_2$ and 0 otherwise. However, we can also define a class of predicates $\mathcal{P} = \{P_x\}_{x \in X}$ where $P_{x_1}(x_2) = 1$ if $x_1 = x_2$ and 0 otherwise. This exponential-size predicate class $\mathcal{P}$ accomplishes the same task as the two-input single predicate. In general, we can take any two-input (or multi-input) predicate and convert it into a predicate class in the same manner.

We give below the definition of (direct access) PPH for a two-input property P. The other definitions follow a similar vein, and are omitted.

Definition 14. *A (direct-access-robust) property-preserving hash family $\mathcal{H} = \{h : X \to Y\}$ for a two-input predicate $P : X \times X \to \{0,1\}$ consists of two efficiently computable algorithms:*

- $\mathcal{H}.\mathsf{Samp}(1^\kappa) \to h$ *is a randomized p.p.t. algorithm that samples a random hash function from $\mathcal{H}$ with security parameter κ.*
- $\mathcal{H}.\mathsf{Eval}(h, y_1, y_2)$ *is a deterministic polynomial-time algorithm that that on input the hash function h and values $y_1, y_2 \in Y$ (presumably $h(x_1)$ and $h(x_2)$ for some $x_1, x_2 \in X$), outputs a single bit.*

Additionally, $h \in \mathcal{H}$ must satisfy the following two properties:

- compressing: $\lceil \log |Y| \rceil < \lceil \log |X| \rceil$, *and*
- direct-access robust: *for any PPT adversary $\mathcal{A}$,*

$$\Pr[h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\kappa);\ (x_1, x_2) \leftarrow \mathcal{A}(h) : P(x_1, x_2) \neq \circledast \wedge \mathcal{H}.\mathsf{Eval}(h(x_1), h(x_2)) \neq P(x_1, x_2)] \leq \mathrm{negl}(\kappa)$$

Any multi-input family of PPH can be converted into a PPH for the corresponding predicate-class with the following simple transformation: $P_{x_1}(x_2) := P(x_1, x_2)$ is transformed into the class of predicates $\{P_x\}_{x \in X}$. In general, single-input PPHs look easier to construct, since the transformed predicate has more information to work with. In fact, we can show an explicit example where the lower bound for the single-predicate version is smaller than the multi-predicate version (see the Gap GreaterThan*proofs in Section 3.4.6).*

Lemma 14. *Let $\mathcal{H}$ be a robust PPH in any model for a two-input predicate P on X. Then, there exists a PPH secure in the same model for the predicate class $\{P_x\}_{x \in X}$ where $P_{x_1}(x_2) = P(x_1, x_2)$.*

Proof. Assume we have a PPH $\mathcal{H}$ for a two-input predicate P and the corresponding predicate class is $\mathcal{P} = \{P_{x_2}\}_{x_2 \in X}$. We will define $\mathcal{H}'$ as follows.

- $\mathcal{H}'.\mathsf{Samp}(1^\kappa) = \mathcal{H}.\mathsf{Samp}(1^\kappa)$.
- $\mathcal{H}'.\mathsf{Eval}(h, y, P'_{x_2}) = \mathcal{H}.\mathsf{Eval}(h, y, h(x_2))$.

Our goal is now to show that an adversary breaking $\mathcal{H}'$ in the security model $\mathcal{H}$ could also break $\mathcal{H}$ in that model.

- Consider the Evaluation-Oracle model. Any evaluation query will be of the form P_{x_1} and x_2, where P_{x_1} has enough information to extract x_1. So, we just query the 2-input Evaluation-Oracle on x_1 and x_2 and pass on the result.

- Consider the Double-Oracle model. Again, any evaluation query will be handled in a similar way, although we get P_{x_1} and y_2, and first need to query for the hash for x_1 and then query the evaluation oracle for $h(x_1), y_2$. Any hash query carries over directly.

- Consider the Direct-Access model. If we are given the code for $\mathcal{H}$, we can easily construct code for $\mathcal{H}'$ and hand an adversary that code with the same distribution as if we were to sample $\mathcal{H}'$ without first sampling $\mathcal{H}$. Thus, if the adversary can break $\mathcal{H}'$ with non-negligible probability, the adversary will break our construction of $\mathcal{H}'$ with the same probability.

□

Here is the full proof that a two-input-predicate PPH implies a PPH for the single-input predicate-class version.

Lemma 15. *Let $\mathcal{H}$ be a robust PPH in any model for a two-input predicate P on X. Then, there exists a PPH secure in the same model for the predicate class $\{P_x\}_{x \in X}$ where $P_{x_1}(x_2) = P(x_1, x_2)$.*

Proof. Assume we have a PPH $\mathcal{H}$ for a two-input predicate P and the corresponding predicate class is $\mathcal{P} = \{P_{x_2}\}_{x_2 \in X}$. We will define $\mathcal{H}'$ as follows.

- $\mathcal{H}'.\mathsf{Samp}(1^\kappa) = \mathcal{H}.\mathsf{Samp}(1^\kappa)$.

- $\mathcal{H}'.\mathsf{Eval}(h, y, P'_{x_2}) = \mathcal{H}.\mathsf{Eval}(h, y, h(x_2))$.

Our goal is now to show that an adversary breaking $\mathcal{H}'$ in the security model $\mathcal{H}$ could also break $\mathcal{H}$ in that model.

- Consider the Evaluation-Oracle model. Any evaluation query will be of the form P_{x_1} and x_2, where P_{x_1} has enough information to extract x_1. So, we just query the 2-input Evaluation-Oracle on x_1 and x_2 and pass on the result.

- Consider the Double-Oracle model. Again, any evaluation query will be handled in a similar way, although we get P_{x_1} and y_2, and first need to query for the hash for x_1 and then query the evaluation oracle for $h(x_1), y_2$. Any hash query carries over directly.

- Consider the Direct-Access model. If we are given the code for $\mathcal{H}$, we can easily construct code for $\mathcal{H}'$ and hand an adversary that code with the same distribution as if we were to sample $\mathcal{H}'$ without first sampling $\mathcal{H}$. Thus, if the adversary can break $\mathcal{H}'$ with non-negligible probability, the adversary will break our construction of $\mathcal{H}'$ with the same probability.

□

3.4 Property Preserving Hashing and Communication Complexity

In this section, we identify and examine a relationship between property-preserving hash families (in the single-input syntax) and protocols in the one-way communication (OWC) model. A OWC protocol is a protocol between two players, Alice and Bob, with the goal of evaluating a certain predicate on their inputs and with the restriction that only Alice can send messages to Bob.

Our first observation is that non-robust property-preserving hash functions and OWC protocols [134] are equivalent except for two changes. First, PPHs require the parties to be computationally efficient, and second, PPHs also require protocols that incur error negligible in a security parameter. It is also worth noting that while we can reference lower-bounds in the OWC setting, these lower bounds are typically of the form $\Omega(n)$ and are not exact. On the other hand, in the PPH setting, we are happy with getting a single bit of compression, and so an $\Omega(n)$ lower bound still does not tell us whether or not a PPH is possible. So, while we can use previously known lower bounds for some well-studied OWC predicates, we need to refine them to be exactly n in the presence of negligible error. We also propose a framework (for total predicates) that yields exactly n lower bounds for $\textsc{Index}_n$, GreaterThan,*and* ExactHamming.

3.4.1 PPH Lower Bounds from One-Way Communication Lower Bounds

In this section, we will review the definition of OWC, and show how OWC lower bounds imply PPH impossibility results.

Definition 15. *[134, 89] A δ-error public-coin OWC protocol Π for a two-input predicate $P : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ consists of a space R of randomness, and two functions $g_a : X_1 \times R \to Y$ and $g_b : Y \times X_2 \times R \to \{0,1\}$ so that for all $x_1 \in X_1$ and $x_2 \in X_2$,*

$$\Pr[r \leftarrow R; y = g_a(x_1; r) : \ g_b(y, x_2; r) \neq P(x_1, x_2)] \leq \delta.$$

A δ-error public-coin OWC protocol Π for a class of predicates *$\mathcal{P} = \{P : \{0,1\}^n \to \{0,1\}\}$, is defined much the same as above, with a function $g_a : X \times R \to Y$, and another function $g_b : Y \times \mathcal{P} \to \{0,1\}$, which instead of taking a second input, takes a predicate from the predicate class. We say Π has δ-error if*

$$\Pr[r \leftarrow R; y = g_a(x; r) : g_b(y, P; r) \neq P(x)] \leq \delta$$

Let $\mathsf{Protocols}_\delta(P)$ denote the set of OWC protocols with error at most δ for a predicate P, and for every $\Pi \in \mathsf{Protocols}_\delta(P)$, let Y_Π be the range of messages Alice sends to Bob (the range of g_a) for protocol Π.

Definition 16. *The randomized, public-coin* OWC complexity *of a predicate P with error δ, denoted $R_\delta^{A \to B}(P)$, is the minimum over all $\Pi \in \mathsf{Protocols}_\delta(P)$ of $\lceil \log |Y_\Pi| \rceil$.*

For a predicate class $\mathcal{P}$, we define the randomized, public-coin OWC complexity with error δ, denoted $R_\delta^{A\to B}(\mathcal{P})$, is the minimum over all $\Pi \in \mathsf{Protocols}_\delta(\mathcal{P})$ of $\lceil \log |Y_\Pi| \rceil$.

A PPH scheme for a two-input predicate[4] P yields a OWC protocol for P with communication comparable to a single hash output size.

Theorem 10. *Let P be any two-input predicate P and $\mathcal{P} = \{P_x\}_{x\in\{0,1\}^n}$ be the corresponding predicate class where $P_{x_2}(x_1) = P(x_1, x_2)$. Now, let $\mathcal{H}$ be a PPH in any model for $\mathcal{P}$ that compresses n bits to $m = \eta n$. Then, there exists a OWC protocol Π such that the communication of Π is m and with negligible error.*

Conversely, the amount of possible compression of any (robust or not) PPH family $\mathcal{H} : \{h : X \to Y\}$ is lower bounded by $R^{A\to B}_{\mathrm{negl}(\kappa)}(P)$. Namely, $\log |Y| \geq R^{A\to B}_{\mathrm{negl}(\kappa)}(\mathcal{P})$.

Essentially, the OWC protocol is obtained by using the public common randomness r to sample a hash function $h = \mathcal{H}.\mathsf{Samp}(1^\kappa; r)$, and then Alice simply sends the hash $h(x_1)$ of her input to Bob. See Appendix 3.4.4 for the proof.

3.4.2 OWC and PPH lower bounds for Reconstructing Predicates

We next leverage this connection together with OWC lower bounds to obtain impossibility results for PPHs. First, we will discuss the total predicate case; we consider some partial predicates in section 3.4.3.

As discussed, to demonstrate the impossibility of a PPH, one must give an explicit n-bit communication complexity lower bound (not just $\Omega(n)$) for negligible error. We give such lower bounds for an assortment of predicate classes by a general approach framework we refer to as reconstructing. Intuitively, a predicate class is reconstructing *if, when given only access to predicates evaluated on an input x, one can, in polynomial time, determine the exact value of x with all but negligible probability.*

Definition 17. *A class $\mathcal{P}$ of total predicates $P : \{0,1\}^n \to \{0,1\}$, is* reconstructing *if there exists a PPT algorithm L (a 'learner') such that for all $x \in \{0,1\}^n$, given randomness r and oracle access to predicates $\mathcal{P}$ on x, denoted $\mathcal{O}_x(P) = P(x)$,*

$$\Pr_r[L^{\mathcal{O}_x}(r) \to x] \geq 1 - \mathrm{negl}(n).$$

Theorem 11. *If $\mathcal{P}$ is a reconstructing class of predicates on input space $\{0,1\}^n$, then a PPH does not exist for $\mathcal{P}$.*

Proof. We will prove this by proving the following OWC lower bound:

$$R^{A\to B}_{\mathrm{negl}(n)}(\mathcal{P}) = n.$$

By Theorem 10, this implies a PPH cannot compress the input and still be correct.

[4]Or rather, for the induced class of single-input predicates $\mathcal{P} = \{P_{x_2}\}_{x_2\in\{0,1\}^n}$, where $P_{x_2}(x_1) = P(x_1, x_2)$; we will use these terminologies interchangeably.

We show that if Alice communicates any fewer than n bits to Bob, then there exists at least one pair of $(x, P) \in \{0,1\}^n \times \mathcal{P}$ such that the probability that the OWC protocol outputs $P(x)$ correctly is non-negligible. Our strategy is to generate pairs (x, P) over some distribution such that, for every fixed choice of randomness of the OWC protocol, the probability that the sampled (x, P) evaluates incorrectly will be $1/\text{poly}(n)$. We first prove that such a distribution violates the negligible-error correctness of OWC.

A bad distribution violates correctness. Let $\mathcal{D}$ be the distribution producing (x, P), r_Π be the randomness of a OWC protocol Π, and $g_b(g_a(x), P) = g_b(g_a(x; r_\Pi), P; r_\Pi)$ for ease of notation. Suppose, for sake of contradiction, that our distribution had a non-negligible chance of producing an error (it is a "bad" distribution), but the correctness of the OWC protocol held. So, we have

$$\begin{aligned}\frac{1}{\text{poly}} &= \Pr_{(x,P)\sim\mathcal{D},r_\Pi}[P(x) \neq g_b(g_a(x), P)] \\ &= \sum_{(x,P)} \Pr_{(x,P)}[\mathcal{D} = (x, P)] \Pr[P(x) \neq g_b(g_a(x), P) | (x, P) = \mathcal{D}],\end{aligned}$$

while the definition of negligible-error OWC protocol states that for every (x, P) pair, $\Pr[P(x) \neq g_b(g_a(x), P)] \leq \text{negl}(n)$. If we plug in $\text{negl}(n)$ for the value of $\Pr[P(x) \neq g_b(g_a(x), P) | (x, P) = \mathcal{D}]$, then we get

$$\Pr_{(x,P)\sim\mathcal{D},r_\Pi}[P(x) \neq g_b(g_a(x), P)] = \text{negl}(n) \cdot \sum_{(x,P)} \Pr_{(x,P)}[\mathcal{D} = (x, P)] = \text{negl}(n).$$

This is a contradiction, and therefore the existence of a distribution producing input-predicate pairs (x, P) that break the OWC protocol with $1/\text{poly}$ probability, violates the (negligible error) correctness.

Generating a bad distribution. So, fix any randomness of the OWC protocol. We will now generate such a distribution $\mathcal{D}$, blind to the randomness of the protocol, that violates correctness of the protocol with $1/\text{poly}(n)$ probability. Let L be the learner for $\mathcal{P}$. We generate this attack as follows:

1. $x \xleftarrow{\$} \{0,1\}^n$.
2. $r \xleftarrow{\$} \mathcal{U}_r$ (to fix the randomness for L).
3. Simulate $L(r)$, answering each query P to $\mathcal{O}_x$ correctly by computing $P(x)$, keeping a list $P_1, \ldots, P_t$ of each predicate query that was *not* answered with $\perp$.
4. $i \xleftarrow{\$} [t]$.
5. Output (x, P_i).

We will show that the probability this attack succeeds will be $\Omega(1/t)$, where t is the total number of queries L makes to $\mathcal{O}_x$. Since L is PPT, with all but negligible probability, $t = \text{poly}(n)$, and therefore the attack succeeds with $1/\text{poly}(n)$ probability.

Note that if Alice and Bob communicate fewer than n bits, for at least half of $x \in \{0,1\}^n$, there exists an x' that maps to the same communicated string: $g_a(x) = g_a(x')$. We analyze the attack success probability via a sequence of steps.

Chose x or x' in a pair. We will first compute the probability that we chose an x that was part of some pair hashing to the same string. Let $Pairs$ be a maximal set of non-overlapping pairs (x, x') that map to the same things. That is for (x, x') and (y, y') in $Pairs$, then none of x, x', y, y' can equal each other. The fraction of elements that show up in $Pairs$ is at least $1/4$. Therefore $\Pr_x[\text{choose } x \text{ or } x' \text{ in a pair}] \geq \frac{1}{4}$.

Chose x and x' that $L(r)$ reconstructs. Now assume that we have chosen either an x or x' in $Pairs$ (that is, fix x and x'). The probability that L distinguishes between x and x' is at least the probability that L correctly reconstructs *both* x and x'. Via a union bound, $\Pr_r[L^{\mathcal{O}_x}(r) = x \wedge L^{\mathcal{O}_{x'}}(r) = x'] \geq 1 - 2\text{negl}(n) = 1 - \text{negl}(n)$.

Chose i that distinguishes x and x'. Next, assume all previous points. Let $i^* \in [t]$ be the first query at which $P_{i^*}(x) \neq P_{i^*}(x')$. Because we fixed r, $L(r)$ now behaves deterministically, although adapts to query inputs, and so P_{i^*} will be the i^*'th query from L to both oracles $\mathcal{O}_x$ and $\mathcal{O}_{x'}$, and must be answered differently ($P_{i^*}(x) \neq P_{i^*}(x')$). Since $g_a(x) = g_a(x')$, we have that $g_b(g_a(x), P_{i^*}) = g_b(g_a(x'), P_{i^*})$ and so either $g_b(g_a(x), P_{i^*}) \neq P_{i^*}(x)$ or $g_b(g_a(x'), P_{i^*}) \neq P_{i^*}(x')$.

The probability we guess $i = i^*$ is $\frac{1}{t}$.

Chose the bad input from x or x'. Assuming all previous points in this list, we get that for one of x or x', the predicate P_i is evaluated incorrectly by g_b. Since we have assumed we chose one of x or x' (uniformly), the probability we chose the x or x' that evaluate incorrectly is $1/2$.

The probability the attack succeeds. Putting all of these points together, after fixing the hash function randomness (and sufficiently large n),

$$\Pr_L[g_b(g_a(x), P_i) \neq P_i(x)] \geq \frac{1}{4} \cdot (1 - \text{negl}(n)) \cdot \frac{1}{t} \cdot \frac{1}{2} \geq \frac{1}{10t}.$$

To recap: we have shown that for every randomness for a OWC protocol, we can produce an input and predicate such that the protocol fails with polynomial-chance. This implies that the OWC protocol does not have negligible error, and furthermore that no PPH can exist for such a predicate class. □

Reconstructing using Index$_n$, GreaterThan, or ExactHamming

We turn to specific examples of predicate classes and sketch why they are reconstructing. For formal proofs, we refer the reader to Appendix 3.4.5.

- *The* $\textsc{Index}_n$ *class of predicates* $\{P_1, \ldots, P_n\}$ *is defined over* $x \in \{0,1\}^n$ *where* $P_i(x) = x_i$*, the i'th bit of* x*.* $\textsc{Index}_n$ *is reconstructing simply because the learner* L *can just query the each of the* n *indices of the input and exactly reconstruct:* $x_i = P_i(x)$*.*

- *The* GreaterThan *class of predicates* $\{P_x\}_{x \in [2^n]}$ *is defined over* $x \in [2^n] = \{0,1\}^n$ *where* $P_{x_2}(x_1) = 1$ *if* $x_1 > x_2$ *and* 0 *otherwise.* GreaterThan *is reconstructing because we can run a binary search on the input space, determining the exact value of* x *in* n *queries.* GreaterThan *is an excellent example for how an adaptive learner* L *can reconstruct.*

- *The* $\textsc{ExactHamming}(\alpha)$ *class of predicates* $\{P_x\}_{x \in \{0,1\}^n}$ *is defined over* $x \in \{0,1\}^n$ *where* $P_{x_2}(x_1) = 1$ *if* $||x_1 - x_2||_0 > \alpha$ *and* 0 *otherwise. To show that* $\textsc{ExactHamming}(n/2)$ *is reconstructing requires a little more work. The learner* L *resolves each index of* x *independently. For each index,* L *makes polynomially many random-string queries* $\mathbf{r}$ *to* $\mathcal{O}_x$*; if the i'th bit of* $\mathbf{r}$ *equals* x_i*, then* $\mathbf{r}$ *is more likely to be within* $n/2$ *hamming distance of* $\mathbf{x}$*, and if the bits are different,* $\mathbf{r}$ *is more likely to not be within* $n/2$ *hamming distance of* $\mathbf{x}$*. The proof uses techniques from [81], and is an example where the learner uses randomness to reconstruct.*

We note that it was already known that $\textsc{Index}_n$ *and* $\textsc{ExactHamming}(n/2)$ *had OWC complexity of* n*-bits for any negligible error [89], though no precise lower bound for randomized OWC protocols was known for* GreaterThan*. What is new here is our unified framework.*

3.4.3 Lower bounds for some partial predicates

In the previous section, we showed how the ability to reconstruct an input using a class of total predicates implied that PPHs for the class cannot exist. This general framework, unfortunately, does not directly extend to the partial-predicate setting, since it is unclear how to define the behavior of an oracle for the predicate. Nevertheless, we can still take existing OWC lower bounds and their techniques to prove impossibility results in this case. We will show that $\textsc{GapHamming}(n, n/2, 1/\sqrt{n})$ *(the promise version of* ExactHamming*) cannot admit a PPH, and that while Gap-k* GreaterThan *has a perfectly correct PPH compressing to* $n - \log(k) - 1$ *bits, compressing any further results in polynomial error (and thus no PPH with more compression).*

First, we define these partial predicates.

Definition 18. *The definitions for* $\textsc{GapHamming}(n, d, \epsilon)$ *and Gap-k* GreaterThan *are:*

- *The* $\textsc{GapHamming}(n, d, \epsilon)$ *class of predicates* $\{P_x\}_{x \in \{0,1\}^n}$ *has* $P_{x_2}(x_1) = 1$ *if* $||x_1 - x_2||_0 \geq d(1+\epsilon)$*,* 0 *if* $||x_1 - x_2||_0 \leq d(1-\epsilon)$*, and* $\circledast$ *otherwise.*

- *The Gap-k* GreaterThan *class of predicates* $\{P_x\}_{x \in [2^n]}$ *has* $P_{x_2}(x_1) = 1$ *if* $x_1 > x_2 + k$*,* 0 *if* $x_1 < x_2 - k$*, and* $\circledast$ *otherwise.*

Now, we provide some intuition for why these lower bounds (and the upper bound) exist.

Gap-Hamming. *Our lower bound will correspond to a refined OWC lower bound for the Gap-Hamming problem in the relevant parameter regime. Because we want to prove that we cannot even compress by a single bit, we need to be careful with our reduction: we want the specific parameters for which we have a lower bound, and we want to know just how the error changes in our reduction.*

Theorem 12. *There does not exist a PPH for* GapHamming$(n, n/2, 1/\sqrt{n})$.

To prove, we show the OWC complexity $R^{A\to B}_{\text{negl}(n)}$ (GapHamming $(n, n/2, 1/\sqrt{n})) = n$. *A* $\Omega(n)$ *OWC lower bound for Gap-Hamming in this regime has been proved in a few different ways [132, 133, 81]. Our proof will be a refinement of [81] and is detailed in appendix 3.4.6.*

The high-level structure of the proof is to reduce Index_n *to* GapHamming *with the correct parameters. Very roughly, the* i*th coordinate of an input* $x \in \{0,1\}^n$ *can be inferred from the bias it induces on the Hamming distance between* x *and random public vectors. The reduction adds negligible error, but since we require* n *bits for negligible-error* Index_n*, we also require* n *bits for a OWC protocol for* GapHamming.

Notice that this style of proof looks morally as though we are "reconstructing" the input x *using* Index_n*. However, the notion of getting a reduction from* Index_n *to another predicate-class in the OWC model is not the same as being able to query an oracle about the predicate and reconstruct based off of oracle queries. Being able to make a similar reconstructing characterization of partial-predicates as we have for total predicates would be useful and interesting in proving more lower bounds.*

Gap-k GreaterThan. *This predicate is a natural extension of* GreaterThan*: we only care about learning that* $x_1 < x_2$ *if* $|x_1 - x_2|$ *is larger than* k *(the gap). Intuitively, a hash function can maintain this information by simply removing the* $\log(k)$ *least significant bits from inputs and directly comparing: if* $h(x_1) = h(x_2)$*, they can be at most* k *apart. We can further remove one additional bit using the fact that we know* x_2 *when given* $h(x_1)$ *(considering Gap-*k *GreaterThan as the corresponding predicate class parameterized by* x_2*).*

For the lower bound, we prove a OWC lower bound, showing $R^{A\to B}_{\text{negl}(n)}(\mathcal{P}) = n - \log(k) - 1$. *This will be a proof by contradiction: if we compress to* $n - \log(k) - 2$ *bits, we obtain many collisions that are more than* $3.5k$ *apart. These far collisions imply the existence of inputs that the OWC protocol must fail on, even given the gap. We are able to find these inputs the OWC must fail on with polynomial probability, and this breaks the all-but-negligible correctness of the protocol. Our formal theorem statement is below.*

Theorem 13. *There exists a PPH with perfect correctness for Gap-*k GreaterThan *compressing from* n *bits to* $n - \log(k) - 1$*. This is tight: no PPH for Gap-*k GreaterThan *can compress to fewer than* $n - \log(k) - 1$ *bits.*

For the proof, see appendix 3.4.6.

3.4.4 Proof of theorem 10: OWC Lower Bounds Imply PPH Lower Bounds

Here is our proof that a lower bound in OWC complexity implies a lower bound for PPHs.

Theorem 14. *Let P be any two-input predicate P and $\mathcal{P} = \{P_x\}_{x\in\{0,1\}^n}$ be the corresponding predicate class where $P_{x_2}(x_1) = P(x_1, x_2)$. Now, let $\mathcal{H}$ be a PPH in any model for $\mathcal{P}$ that compresses n bits to $m = \eta n$. Then, there exists a OWC protocol Π such that the communication of Π is m and with negligible error.*

Proof. Let P'_x be the transformed predicate for P_x, so $P'_x(y_1) = \mathcal{H}.\mathsf{Eval}(h_r, y_1, P)$. Π will operate as follows:

- Alice computes $g_a(x_1; r) = h_r(x_1)$ where $h_r = \mathcal{H}.samp(1^\kappa; r)$ (runs the sampling algortihm with public randomness r).
- Bob computes $g_b(y, x_2; r) = \mathcal{H}.\mathsf{Eval}(h_r, y_1, P_{x_2})$ where Bob can also evaluate $h_r = \mathcal{H}.\mathsf{Samp}(1^\kappa; r)$ with the public randomness and can compute $P'_{x_2}(y_1) = \mathcal{H}.\mathsf{Eval}(h_r, y_1, P_{x_2})$.

First, the communication of Π is clearly m bits since Alice only sends a single hashed value of x_1 during the protocol.

Second, Π is correct with all but negligible probability. This follows directly from the soundness or correctness of the PPH — even a non-robust PPH has correctness with overwhelming probability. Formally, for any two inputs from Alice and Bob, x_1 and x_2 respectively,

$$\begin{aligned}&\Pr_r[g_b(g_a(x_1; r), x_2; r) = P(x_1, x_2)]\\ &\quad = \Pr_{h_r \leftarrow \mathcal{H}.\mathsf{Samp}(1^\kappa)}[P'_{x_2}(h_r(x_1)) = P(x_1, x_2)] \geq 1 - \mathrm{negl}(n).\end{aligned}$$

□

3.4.5 Proofs that Index$_n$, GreaterThan, and ExactHammingare Reconstructing

First, we will go over Index$_n$. *It was already known that* Index$_n$ *had OWC complexity of n-bits for any negligible error [89]. While the methods of Kremer et. al. give a lower bound relative to the error, we care about negligible error from our definition of PPHs.*

Lemma 16. Index$_n$ *is reconstructing.*

Proof. The learning algorithm L is straightforward: for every $\mathbf{x} \in \{0,1\}^n$, $L^{O_\mathbf{x}}$ makes n static queries $P_1, \ldots, P_n$ where $P_j(x) = x_j$. After n queries, L has $(x_1, \ldots, x_n) = \mathbf{x}$. Note that L does not require adaptivity or randomness. □

Corollary 3. *There does not exist a PPH for* Index$_n$.

Now we will examine GreaterThan. GreaterThan *is a problem where the deterministic lower bound is known to be exactly n, but no precise lower bound for randomized OWC protocols is known. Recall that for equality, we have the same deterministic lower bound, but a randomized protocol with negligible error can have significantly smaller OWC complexity $O(\lambda)$. The same will* not *be true of* GreaterThan.

Lemma 17. GreaterThan *is reconstructing.*

Proof-sketch. For every $x \in [2^n]$, $L^{\mathcal{O}_x}$ is simply binary searching for x using the greater-than predicate. So, the first query is $\mathcal{O}_x(2^{n-1})$ and depending on the answer, the next query is either 2^{n-2} or $2^{n-1} + 2^{n-2}$, and so forth. There are a total of n queries, and from those queries L can exactly reconstruct x. □

Corollary 4. *There does not exist a PPH for* GreaterThan

Next, we turn to ExactHamming*, with parameter α.*

Definition 19. *The* $\text{ExactHamming}_\alpha$ *two-input predicate is defined as*

$$\text{ExactHamming}_\alpha(x_1, x_2) = \begin{cases} 0 & \text{if } ||x_1 - x_2||_0 \leq \alpha \\ 1 & \text{if } ||x_1 - x_2||_0 > \alpha \end{cases}$$

While making the claim that ExactHamming *has OWC complexity of n bits follows from Theorem 12 in the following section, we are able to demonstrate the flexibility of reconstructing predicates; the proof of this lemma uses an L that is randomized.*

Lemma 18. $\text{ExactHamming}(n/2)$ *is reconstructing.*

Proof. This proof borrows techniques from [81], where they showed that GapHamming $(n/2, c\sqrt{n})$ required $\Omega(n)$ bits of communication, by reducing Index_n to an instance of this problem. We will have L use $\mathcal{O}_{\mathbf{x}}$ to create this same GapHamming instance just as Alice and Bob separately computed it.

$L^{\mathcal{O}_{\mathbf{x}}}$ will use the following algorithm:

1. For every $i \in [n]$ and $j \in [m]$:
   (a) Use the randomness to generate a new random vector $\mathbf{r}_{i,j} \xleftarrow{\$} \{0,1\}^n$.
   (b) Let $b_{i,j} \leftarrow \mathbf{r}_{i,j}[i]$ and $a_{i,j} = 1 - \mathcal{O}_{\mathbf{x}}(\mathbf{r}_{i,j})$.
2. For every $i \in [n]$, let $\mathbf{x}'_i = (a_{i,1}, \ldots, a_{i,m})$ and $\mathbf{y}'_i = (b_{i,1}, \ldots, b_{i,m})$.
3. For every $i \in [n]$, let $\hat{x}_i = 1$ if $||\mathbf{y}'_i - \mathbf{x}'_i||_0 \leq n/2$ and $\hat{x}_i = 0$ otherwise.
4. Return $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_n)$.

This algorithm is exactly the algorithm Alice and Bob use in the proof that GapHamming requires n bits of communication in appendix 3.4.6: L acts out Alice's part by using $\mathcal{O}_{\mathbf{x}}$ to compute exact-hamming between $\mathbf{r}_{i,j}$ and $\mathbf{x}$, and acts out Bob's part by just taking the i'th coordinate from that random vector as the guess for the i'th bit of $\mathbf{x}$. Now, without generality assume n is odd, and the analysis is then the same:

- Assume $x_i = 1$. Then, $\mathrm{E}_{\mathbf{r}_{i,j}}[||\mathbf{x}'_i - \mathbf{y}'_i||_0] \leq \frac{n}{2} - \frac{\sqrt{2\pi}}{e^2}\sqrt{n}$, and so as long as $m = O(n^2)$, a Chernoff bound yields $\Pr[||\mathbf{x}'_i - \mathbf{y}'_i||_0 \geq \frac{n}{2}] \leq e^{-O(n)} = \mathrm{negl}(n)$. And so, the probability that we guess x_i is 0 when $x_i = 1$ is negligible.

- Assume $x_i = 0$. We have $\mathrm{E}_{\mathbf{r}_{i,j}}[||\mathbf{x}'_i - \mathbf{y}'_i||_0] \leq \frac{n}{2} + \frac{\sqrt{2\pi}}{e^2}\sqrt{n}$. Again, as long as $m = O(n^2)$, a Chernoff bound yields $\Pr[||\mathbf{x}'_i - \mathbf{y}'_i||_0 \leq \frac{n}{2}] \leq e^{-O(n)} = \mathrm{negl}(n)$.

And with that, the chance that we guess x_i incorrectly is negligible. □

Corollary 5. *There does not exist a PPH for* ExactHamming$(n/2)$.

3.4.6 Proofs of Lower bounds for Gap-Hamming and Gap-GreaterThan

Proof that Gap-Hamming Requires n bits of Communication

Here is the full proof that Gap-Hamming Requires n bits of Communication. Recall the definition of the GapHamming *problem.*

Definition 20. *The* GapHamming(n, d, ϵ) *promise predicate acts on a pair of vectors from* $\{0,1\}^n$ *and is defined as follows.*

$$P(\mathbf{x}_1, \mathbf{x}_2) = \begin{cases} 0 & \text{if } ||\mathbf{x}_1 \oplus \mathbf{x}_2||_0 \leq d(1-\epsilon) \\ 1 & \text{if } ||\mathbf{x}_1 \oplus \mathbf{x}_2||_0 \geq d(1+\epsilon) \\ \circledast & \text{otherwise} \end{cases}$$

where $||\cdot||_0$ *denotes the* ℓ_0 *norm, or equivalently Hamming weight.*

Theorem 15. *The randomized OWC complexity of* GapHamming$_n(n/2, \sqrt{n}/2)$ *with negligible error is exactly n;*

$$R^{A\to B}_{\mathrm{negl}(n)}(\textsc{GapHamming}(n, n/2, 1/\sqrt{n})) = n.$$

Proof. This will be a randomized reduction of Index$_n$ to GapHamming for an arbitrary error δ; we will show that this randomized reduction introduces negligible error, and so if we want negligible error for GapHamming on these parameters, we require $\delta = \mathrm{negl}(n)$, too. We will take Alice's input $\mathbf{x}$ and Bob's index $i \in [n]$ and create two new vectors, $\mathbf{a}$ and $\mathbf{b}$ n-bit vectors, correlated using the public randomness so that if $x_i = 1$, $\mathbf{a}$ and $\mathbf{b}$ will be within $n/2 - \sqrt{n}/2$ distance from each other and if $x_i = 0$, the vectors will be at least $n/2 + \sqrt{n}/2$ distance with all but negligible probability (over n).

Without loss of generality, assume n is odd.

- For each coordinate b_j, Bob samples the same public randomness $\mathbf{r}_j \leftarrow \{0,1\}^n$ and sets $b_j \leftarrow r_j$. Bob is essentially pretending $\mathbf{r}_j$ is Alice's vector.

- For each coordinate a_j, Alice samples the public randomness $\mathbf{r}_j \xleftarrow{\$} \{0,1\}^n$. If $||\mathbf{x} - \mathbf{r}_j||_0 < n/2$, she sets $a_j \leftarrow 1$, and if $||\mathbf{x} - \mathbf{r}_j||_0 > n/2$, she sets $a_j \leftarrow 0$. Alice is marking if $\mathbf{r}_j$ is a good proxy for $\mathbf{x}$.

We now need to argue that $\mathbf{a}$ and $\mathbf{b}$ are close if $x_i = 1$ and far otherwise. We will do this by computing the expected hamming distance between $\mathbf{a}$ and $\mathbf{b}$, and then applying a Chernoff bound. Let's go through both cases.

- Assume $x_i = 1$. Then, $\mathrm{E}_{\mathbf{r}}[||\mathbf{a} - \mathbf{b}||_0] = \sum_{j=1}^{n} \Pr_{\mathbf{r}}[a_j \neq b_j]$. Now, looking at $\Pr_{\mathbf{r}}[a_j \neq b_j]$, we have the two more cases. Either $\mathbf{x}$ and $\mathbf{r}_j$ agree on *strictly* less than or greater than $(n-1)/2$ bits (meaning r_i is not used in determining a_j); or, $\mathbf{x}$ and $\mathbf{r}_j$ agree on exactly $(n-1)/2$ bits. In the second case, since $x_i = 1$, the probability that $b_j = a_j$ is 1. So,

$$\Pr_{\mathbf{r}}[a_j \neq b_j] = \Pr[\text{Case 1}] \cdot \frac{1}{2} - \Pr[\text{Case 2}] \cdot 0$$

  Using Stirling's approximation, we get that $\Pr[\text{Case 2}] = \frac{c\sqrt{2}}{\sqrt{n-1}}$, where $\frac{2\sqrt{\pi}}{e^2} \leq c \leq \frac{e}{\pi\sqrt{2}}$. And so,

$$\begin{aligned}\Pr_{\mathbf{r}}[a_j = b_j] &= \left(1 - \frac{c\sqrt{2}}{\sqrt{n-1}}\right) \cdot \frac{1}{2} \\ &\leq \frac{1}{2} - \frac{c}{\sqrt{2n}}\end{aligned}$$

  Now, when we commute the expected hamming distance if $x_i = 1$, we get

$$\mathrm{E}_{\mathbf{r}}[||\mathbf{a} - \mathbf{b}||_0] = \sum_{j=1}^{n} \Pr_{\mathbf{r}}[a_j \neq b_j] \leq n \cdot (\frac{1}{2} - \frac{c}{\sqrt{2n}}) = \frac{n}{2} - \frac{c\sqrt{n}}{\sqrt{2}}$$

  Plugging in the lower bound for c we computed with Stirling's approximation, we have

$$\mathrm{E}_{\mathbf{r}}[||\mathbf{a} - \mathbf{b}||_0] \leq \frac{n}{2} - \frac{\sqrt{2\pi}}{e^2} \cdot \sqrt{n}$$

  Now, using a Chernoff bound, we get that $\Pr[||\mathbf{a} - \mathbf{b}||_0 > \frac{n}{2} + \frac{\sqrt{n}}{4}] \leq e^{-O(n)} = \mathrm{negl}(n)$.

- Assume $x_i = 0$. We will use the same analysis as before, but now in the second case, we have $x_i = 0$, so the probability that $r_i = a_j$ is 0. And hence,

$$\mathrm{E}_{\mathbf{r}}[||\mathbf{a} - \mathbf{b}||_0] \geq \frac{n}{2} + \frac{\sqrt{2\pi}}{e^2} \cdot \sqrt{n}$$

  Again, using a Chernoff bound, we get that $\Pr[||\mathbf{a} - \mathbf{b}||_0 < \frac{n}{2} + \frac{\sqrt{n}}{4}] \leq e^{-O(n)} = \mathrm{negl}(n)$.

Therefore, with all but negligible probability in n, this randomized reduction is correct. □

Proof that Gap-k Greater-Than requires $n - \log(k) - 1$ bits

First, let us recall the definition of the Gap-k greater-than predicate P.

$$P_{x_1}(x_2) = \begin{cases} 1 & \text{if } x_1 > x_2 \text{ and } |x_1 - x_2| > k \\ 0 & \text{if } x_1 \leq x_2 \text{ and } |x_1 - x_2| > k \\ * & \text{otherwise} \end{cases}$$

Theorem 16. *For $k \geq 1$, the OWC complexity of gap-k* GreaterThan *is*

$$R^{A \to B}_{\text{negl}(n)}(\mathcal{P}) = n - \log(k) - 1$$

and in fact there exists a protocol compressing by $\log(k) + 1$ bits that has no error.

Proof. Assume k is a power of 2. All other k follow: we will be unable to compress by more than $\lceil \log(k) \rceil + 1$ bits. Let P be the Gap-k GreaterThan predicate.

Designing a PPH. The proof that we can compress by $\log(k) + 1$ is simply that our hash function h just removes the last $\log(k) + 1$ bits. However, P' must do a little work:

- Let $L = h^{-1}(h(x)) = \{x_0, x_1, \ldots, x_{2k}\}$ be the list, in order, of all elements mapping to $h(x)$, where $x_0 + 2k = x_1 + 2k - 1 = \cdots = x_{2k}$.
- If $a \leq x_k$, $P'_a(h(x)) = 0$, and if $a > x_k$, $P'_a(h(x)) = 1$.

First we show this algorithm is correct. For every $x, a \in [2^n]$, if $P_a(h(x)) = 0$, then $a \leq \min h^{-1}(h(x)) + k \leq x + k$. If $a < x$, then $P_a(h(x))$ answers correctly, but if $a > x$, we get that $|x - a| \leq k$, and so our output is still alright since a is within the gap around x. Similarly, if $P_a(h(x)) = 1$, then $a > \max h^{-1}(h(x)) - k \geq x - k$. For the same reasons the output is either correct or within the gap.

Lower bound. Now we want to show that if h compresses by more than $\log(k)+1$ bits, we can non-adaptively find two inputs such that $P'_a(h(x)) \neq P_a(x)$ with non-negligible probability. In fact we will show that we can guess a, x with probability at least $\frac{1}{400n}$. Our method will be first to guess an x that is "bad" (collides with an x' more than $\frac{5}{2}k$ from it), guess if it is smaller or bigger than x' (b), and then guess by how much x' is smaller or bigger than x ($2^{s-1} \leq |x - x'| \leq 2^s$):

1. $x \xleftarrow{\$} [N]$
2. $b \xleftarrow{\$} \{0, 1\}$ and $s \xleftarrow{\$} \{\log k + 1, \ldots, n\}$.
3. If $b = 0$: $a \xleftarrow{\$} \{x - 2^s, \ldots, x - (k + 1)\}$ and output x and a.
   If $b = 1$: $a \xleftarrow{\$} \{x + (k + 1), \ldots, x + 2^s\}$ and output x and a.

Let h be any function hashing n bits to $n - (\log(k) + 2)$ bits (again assume k is a power of 2). Let $K = 3.5k$, we want to bound the probability we choose a random

input x and end up with $h(x)$ having a pre-image size at least size K:

$$\begin{aligned}\Pr_{x\xleftarrow{\$}[N]}\left[|h^{-1}(h(x))| \geq K\right] &= 1 - \Pr_{x\xleftarrow{\$}[N]}\left[|h^{-1}(h(x))| < K\right] \\ &\geq 1 - (K-1)(2^{n-\log(k)-1} - 1)2^{-n} \\ &\geq 1 - (K-1)(\frac{1}{4k}) \\ &\geq 1 - \frac{3.5k}{4k} = \frac{1}{8}\end{aligned}$$

Consider any preimage $h^{-1}(y)$ of size at least $3.5k$: if we sort the set $h^{-1}(y)$, then pair off the first x in the sorted list with the $\frac{5}{2}\cdot k$'th, the second with the $\frac{5}{2}k+1$'th and so on, then choosing $x \xleftarrow{\$} h^{-1}(y)$, with probability $\frac{4}{7}$, x will have an x' it is paired with. For all a in between x and x', and at least distance k from *both* of them, $P'_a(h(x))$ is wrong more often than $P'_a(h(x'))$ or vice-versa. For each x, x' pair, consider all a in between x and x' and at least distance k from both of them: $P'_a(h(x)) = P'_a(h(x'))$. So, for one of x or x', this will evaluate incorrectly. Therefore, one of x or x' will have that, for at least half of the a's in between and distance k from both, P'_a evaluates incorrectly. We also have that since x and x' are at least $\frac{5}{2}k$ apart, there are at least $\frac{k}{2}$ elements a that are distance k from both of them. If we choose at random from elements at least distance k from x, we get the probability of choosing an a at distance k from both x and x' is $\frac{1}{3}$.

We will call an x *bad* if it has an x' such that $h(x) = h(x')$ and $|x - x'| \geq 3.5k$ (which we call 'paired'), and for all the a in between x and x' and distance k from both, more than half of them evaluate incorrectly on x.

$$\begin{aligned}\Pr_x[x \text{ is bad}] &\geq \Pr[x \text{ is bad}||h^{-1}(h(x))| \geq K] \cdot \Pr[|h^{-1}(h(x))| \geq K] \\ &\geq \Pr[x \text{ is bad}||h^{-1}(h(x))| \geq K \wedge \exists \text{ paired } x'] \cdot \Pr[\exists \text{ paired } x'||h^{-1}(h(x))| \geq K] \cdot \frac{1}{8} \\ &\geq \Pr[x \text{ is bad}||h^{-1}(h(x))| \geq K \wedge \exists \text{ paired } x' \wedge x \text{ has more incorrect } a] \\ &\quad \cdot \frac{1}{2} \cdot \frac{4}{7} \cdot \frac{1}{8} \\ &\geq \frac{1}{28}\end{aligned}$$

Now, assume that we have chosen a bad x. We will compute the probability we choose an a that $P_a(x) \neq P'_a(h(x))$. Consider x and its pair x': x is wrong on at least half of the a between x and x', so our goal is to sample between x and x' without knowing x'. First, we guess whether $x < x'$ or vice-versa (our choice of the bit b). Then, we guess how far apart they are to the nearest power of 2 (our choice of $s \in [n]$). Finally, if we have guessed both of these correctly, we sample in the range $x \pm s$, and with probability at least $1/2$ we are sampling in the range (x, x'), and again with probability at least $1/2$, we sample an a that evaluates incorrectly for x. Formally, we have:

- Assume x is bad, and so is paired with an x'. $\Pr_{x,b,s,a}[P'_a(h(x)) \neq P_a(x)] \geq \Pr_{x,b,s,a}[P'_a(h(x)) \neq P_a(x)|b \text{ is correct}] \cdot \frac{1}{2}$.

- Now assume that both x is bad and b is chosen correctly (that is, we know $x < x'$ or $x' < x$). $\Pr_{x,b,s,a}[P'_a(h(x)) \neq P_a(x)] \geq \Pr_{x,b,s,a}[P'_a(h(x)) \neq P_a(x) | 2^{s-1} < |x - x'| \leq 2^s] \cdot \frac{1}{n}$.

- Assume x is bad, b is chosen correctly, and s is also guessing the range between x and x' correctly. We have $\Pr_{x,b,s,a}[P'_a(h(x)) \neq P_a(x)] \geq \Pr_{x,b,s,a}[P'_a(h(x)) \neq P_a(x) | a \in (x, x')] \cdot \frac{1}{2}$ since $\Pr_{x,b,s,a}[a \in (x, x')] \geq \frac{1}{2}$ given $a \in (x, x + 2^s)$ or $a \in (x - 2^s, x)$ when $b = 0$ or $b = 1$ respectively.

- Assume we have chosen an $a \in (x, x')$ or (x', x) (whichever is correct). The probability that $|x - a|$ and $|x' - a| > k$ is at least $\frac{1}{3}$ (since we guarantee that a is at least distance k from x).

- Finally, assume all of the previous points. We get $\Pr_{x,b,s,a}[P'_a(h(x)) \neq P_a(x)] \geq \frac{1}{2}$ because x is bad and we are choosing $a \in (x, x')$ or (x', x) (whether $b = 0$ or 1) where a is distance more than k from both x and x'. So, $P'_a(x)$ will evaluate incorrectly on at least half of all such a's.

- Putting all of these conditionals together (multiplying them), we have

$$\Pr_{x,b,s,a}[P'_a(h(x)) \neq P_a(x)] \geq \frac{1}{28} \cdot \frac{1}{2} \cdot \frac{1}{n} \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{336n}.$$

This completes the proof: if we try to compress more than $\lceil \log(k) \rceil + 1$ bits, we end up being able to use the above attack to find a bad input on the hash function with probability at least $\frac{1}{400n}$. □

A Different Lower Bound for Two-input Greater-Than. *Here we will intuitively describe why the lower bound for Two-Input Greater-Than is $n - \log(k)$, one bit less than the lower bound on the single-input version. Consider the construction for a single-input gap-k Greater-Than PPH, as described in the proof above. $h(x)$ removes the last $\log(k)+1$ bits from x, and we are able to compare a value a to $h^{-1}(x)$, checking if a is in the lower half or the higher half. If we instead are only given $h(x)$ and $h(a)$, we can only check that a is in the $h^{-1}(x)$, and have no sense of where.*
So, in the two-input case, we have a simple upper bound: our hash $h'(x)$ removes exactly the $\log(k)$ lowest bits, and $P'(h(x), h(a))$ simply compare $h(x) > h(a)$. The proof that this is optimal follows the same structure as above, except we let $K = 2k$. Now, if our adversary finds a random collision, $h(x) = h(a)$, there is a large enough chance that $|x - a| > k$, which would violate the correctness of the PPH.

3.4.7 A Gap-Hamming PPH from Collision Resistance

Our first construction is a robust m/n-compressing GapHamming(n, d, ϵ) *PPH for any $m = n^{\Omega(1)}$, $d = o(n/\log \kappa)$ and any constant $\epsilon > 0$. Security of the construction holds under the (standard) assumption that collision-resistant hash function families (CRHFs) exist.*

We now informally describe the idea of the construction which, in one word, is "subsampling". In slightly more detail, the intuition is to notice that if $\mathbf{x}_1 \in \{0,1\}^n$ *and* $\mathbf{x}_2 \in \{0,1\}^n$ *are* close*, then* most small enough subsets *of indices of* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *will match identically. On the other hand, if* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *are* far*, then most* large enough subsets *of indices will differ. This leads us to the first idea for the construction, namely, fix a collection of sets* $\mathcal{S} = \{S_1, \ldots, S_k\}$ *where each* $S_i \subseteq [n]$ *is a subset of appropriately chosen size* s*. On input* $\mathbf{x} \in \{0,1\}^n$*, output* $\mathbf{y} = (\mathbf{x}|_{S_1}, \ldots, \mathbf{x}|_{S_k})$ *where* $\mathbf{x}|_S$ *denotes the substring of* $\mathbf{x}$ *indexed by the set* S*. The observation above tells us that if* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *are close (resp. far), so are* $\mathbf{y}_1$ *and* $\mathbf{y}_2$*.*

However, this does not compress the vector $\mathbf{x}$*. Since the union of all the sets* $\bigcup_{i\in[k]} S_i$ *has to be the universe* $[n]$ *(or else, finding a collision is easy), it turns out that we are just comparing the vectors index-by-index. Fortunately, it is not necessary to output* $\mathbf{x}|_{S_i}$ *by themselves; rather we can simply output the collision-resistant hashes. That is, we will let the PPH hash of* $\mathbf{x}$*, denoted* $\mathbf{y}$*, be* $(g(\mathbf{x}|_{S_1}), \ldots, g(\mathbf{x}|_{S_k}))$ *where* g *is a collision resistant hash function randomly drawn from a CRHF family.*

This simple construction works as long as s*, the size of the sets* S_i*, is* $\Theta(n/d)$*, and the collection* $\mathcal{S}$ *satisfies that any subset of disagreeing input indices* $T \subseteq [n]$ *has nonempty intersection with roughly the corresponding fraction of subsets* S_i*. The latter can be achieved by selecting the* S_i *of size* $\Theta(n/d)$ *at random, or alternatively as defined by the neighbor sets of a bipartite expander. We are additionally constrained by the fact that the CRHF must be secure against adversaries running in time* $\mathrm{poly}(\kappa)$*. So, let* $t = t(\kappa)$ *be the smallest output size of the CRHF such that it is* $\mathrm{poly}(\kappa)$*-secure. Since the input size* s *to the CRHF must be* $\omega(t)$ *so that* g *actually compresses, this forces* $d = o(n/t)$*.*

Before presenting our construction more formally, we define our tools.

- *We will use a family of CRHFs that take inputs of variable size and produce outputs of* t *bits and denote it by* $\mathcal{H}_t = \{h : \{0,1\}^* \to \{0,1\}^t\}$*. We implicitly assume a procedure for sampling a seed for the CRHF given a security parameter* 1^κ*. One could set* $t = \omega(\log \kappa)$ *and assume the exponential hardness of the CRHF, or set* $t = \kappa^{O(1)}$ *and assume polynomial hardness. These choices will result in different parameters of the PPH hash function.*

- *We will use an* $(n, k, D, \gamma, \alpha)$-bipartite expander $G = (L \cup R, E)$ *which is a* D*-left-regular bipartite graph, with* $|L| = n$ *and* $|R| = k$ *such that for every* $S \subset L$ *for which* $|S| \leq \gamma n$*, we have* $|N(S)| \geq \alpha|S|$*, where* $N(S)$ *is the set of neighbors of* S*. For technical reasons, we will need the expander to be* δ-balanced *on the right, meaning that for every* $v \in R$*,* $|N(v)| \geq (1-\delta)nD/k$*.*

  A simple probabilistic construction shows that for every $n \in \mathbb{N}$*,* $k = o(n)$ *and constant* $a \in (0,1)$*, and any* $\gamma = o(\frac{k}{n\log(n/k)})$ *and* $D = \Theta(\log(1/\gamma))$ *so that for every* $\delta > 0$*,* δ*-balanced* $(n, k, D, \gamma, \alpha)$*-bipartite expanders exist. In fact, there are even explicit efficient constructions that match these parameters [34]. The formal lemma statement and proof are given in Appendix 3.4.8.*

We next describe the general construction, and then discuss explicit parameter settings and state our formal theorem.

Robust GapHamming(n, d, ϵ) PPH family $\mathcal{H}$ from any CRHF

Our (n, m, d, ϵ)-robust PPH family $\mathcal{H} = (\mathcal{H}.\mathsf{Samp}, \mathcal{H}.\mathsf{Eval})$ is defined as follows.

- $\mathcal{H}.\mathsf{Samp}(1^\kappa, n)$. Fix a δ-balanced $(n, k, D, \gamma, \alpha)$-bipartite expander $G = (L \cup R, E)$ (either deterministically or probabilistically). Sample a CRHF $g \leftarrow \mathcal{H}_t$. Output $h = (G, g)$.

- $\mathcal{H}.\mathsf{Hash}(h = (G, g), \mathbf{x})$. For every $i \in [k]$, compute the (ordered) set of neighbors of the i-th right vertex in G, denoted $N(i)$. Let $\hat{\mathbf{x}}^{(i)} := \mathbf{x}|_{N(i)}$ be $\mathbf{x}$ restricted to the set $N(i)$. Output

$$h(\mathbf{x}) := \Big(g(\hat{\mathbf{x}}^{(1)}), \ldots, g(\hat{\mathbf{x}}^{(k)})\Big)$$

  as the hash of $\mathbf{x}$.

- $\mathcal{H}.\mathsf{Eval}(h = (G, g), \mathbf{y}_1, \mathbf{y}_2)$. Compute the threshold $\tau = D \cdot d \cdot (1 - \epsilon)$. Parse $\mathbf{y}_1 = (\hat{\mathbf{y}}_1^{(1)}, \ldots, \hat{\mathbf{y}}_1^{(k)})$ and $\mathbf{y}_2 = (\hat{\mathbf{y}}_2^{(1)}, \ldots, \hat{\mathbf{y}}_2^{(k)})$. Compute

$$\Delta' = \sum_{i=1}^{k} \mathbb{1}(\hat{\mathbf{y}}_1^{(i)} \neq \hat{\mathbf{y}}_2^{(i)}),$$

  where $\mathbb{1}$ denotes the indicator predicate. If $\Delta' \leq \tau$, output ***CLOSE***. Otherwise, output ***FAR***.

Table 3.1: Construction of a robust GapHamming(n, d, ϵ) PPH family from CRHFs.

Setting the Parameters. *The parameters required for this construction to be secure and constructible are as follows.*

- *Let $n \in \mathbb{N}$ and constant $\epsilon > 0$.*
- *We require two building blocks: a CRHF and an expander. So, let $\mathcal{H}_t = \{g : \{0,1\}^* \to \{0,1\}^t\}$ be a family of CRHFs secure against $\text{poly}(\kappa)$-time adversaries. Let G be a δ-balanced $(n, k, D, \gamma, \alpha)$-expander for a constant δ bounding the degree of the right-nodes, where n is the size of the left side of the graph, k is the size of the right, D is the left-degree, γn is the upper bound for an expanding set on the right that expands to a set of size α times the original size.*
- *These building blocks yield the following parameters for the construction: compression is $\eta = kt/n$ and our center for the Gap-Hamming problem is bounded by $\frac{\gamma n}{(1+\epsilon)} \leq d < \frac{k}{D(1-\epsilon)}$.*

If we consider what parameter settings yield secure CRHFs with output size t and for what parameters we have expanders, we have a PPH construction where given any n and ϵ, there exists a $d = o(n)$ and $\eta = O(1)$ such that Construction 3.1 is a robust PPH for gap-Hamming. We will see that the smaller t is, the stronger the CRHF security assumption, but the better our compression.

Now, given these settings of parameters, we will formally prove Construction 3.1 is a robust Gap-Hamming PPH.

Theorem 17. *Let κ be a security parameter. Assuming that exponentially secure CRHFs exist, for any polynomial $n = n(\kappa)$, and any constants $\epsilon, \eta > 0$, Construction 3.1 is an η-compressing robust property preserving hash family for $\text{GapHamming}(n, d, \epsilon)$ where $d = o(n/\log \kappa \log\log \kappa)$. Assuming only that polynomially secure CRHFs exist, for any constant $c > 0$, we achieve $d = o(n/\kappa^c)$.*

Proof. Before getting into the proof, we more explicitly define the parameters ti include parameters associated with the expander in our construction. Once we have defined these parameters, we will show how each of these parameters is used to prove the construction is correct and robust. So, in total, we have the following paramters and implied constraints for our construction (including the graph):

1. Let $n \in \mathbb{N}$ be the input size and let $\epsilon > 0$ be any constant.
2. Our CRHF is $\mathcal{H}_t = \{g : \{0,1\}^* \to \{0,1\}^t\}$.
3. Our expander will be a δ-balanced $(n, k, D, \gamma, \alpha)$-expander, where $k < n/t$, $\gamma = o(\frac{k}{n \log(n/k)})$, $D = \Theta(\log(1/\gamma))$, and $\alpha > D \cdot \frac{d(1-\epsilon)}{\gamma n}$.
4. Our center for the gap-hamming problem is d, and is constrained by $\frac{\gamma n}{1+\epsilon} \leq d < \frac{k}{D(1-\epsilon)}$.
5. Constraint 4 implies that $k = n^{\Omega(1)}$, since $\frac{\gamma n \cdot D(1-\epsilon)}{1+\epsilon} < k$.

Now, we prove our construction is well-defined and efficient. Fix any $\delta, a \in (0,1)$. Using lemma 19, proved in the appendix, we know that δ-balanced $(n, k, D, \gamma, \alpha = an)$-bipartite expanders exist and can be efficiently sampled for $k = o(n/\log n)$, $D = \Theta(\log\log n)$, and $\gamma = \tilde{\Theta}(1/\log n)$. Thus, sampling G before running the construction is efficient. Once we have a G, sampling and running a CRHF $k = O(n)$ times is efficient. Comparing k outputs of the hash function is also efficient. Therefore, each of $\mathcal{H}.\mathsf{Samp}$, $\mathcal{H}.\mathsf{Hash}$, and $\mathcal{H}.\mathsf{Eval}$ is efficient in $\kappa = \text{poly}(n)$.

Now, we prove that Construction 3.1 is compressing. Points 2 and 3 mean that $m = k \cdot t < n/t \cdot t = n$, as required.

Lastly, we will prove our construction is robust. Let $\mathcal{A}$ be a PPT adversary. We will show that $\mathcal{A}$ (in fact, even an unbounded adversary) cannot find $\mathbf{x}_1$ and $\mathbf{x}_2$ such that $||\mathbf{x}_1 - \mathbf{x}_2|| \leq d(1-\epsilon)$ but $\mathcal{H}.\mathsf{Eval}(h, h(\mathbf{x}_1), h(\mathbf{x}_2))$ evaluates to $\boldsymbol{FAR}$, and that $\mathcal{A}$ must break the collision-resistance of $\mathcal{H}_t$ in order to find $\mathbf{x}_1$ and $\mathbf{x}_2$ where $||\mathbf{x}_1 - \mathbf{x}_2|| \geq d(1+\epsilon)$ but $\mathcal{H}.\mathsf{Eval}(h, h(\mathbf{x}_1), h(\mathbf{x}_2))$ evaluates to $\boldsymbol{CLOSE}$.

- First, consider any $\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ where $||\mathbf{x}_1 - \mathbf{x}_2||_0 \leq d(1-\epsilon)$. Let $\Delta = ||\mathbf{x}_1 - \mathbf{x}_2||_0$. So, consider the set $S \subset L$ corresponding to the indices that are different between $\mathbf{x}_1$ and $\mathbf{x}_2$, and $T = N(S) \subset R$. The maximum size of T is $|S| \cdot D$, the degree of the graph.

  For every $i \in T$, we get that the intermediate computation has $\hat{\mathbf{x}}_1^{(i)} \neq \hat{\mathbf{x}}_2^{(i)}$, but for every $j \notin T$, we have $\hat{\mathbf{x}}_1^{(j)} = \hat{\mathbf{x}}_2^{(j)}$ which implies $\hat{\mathbf{y}}_1^{(j)} = \hat{\mathbf{y}}_2^{(j)}$ after applying g. Therefore $\sum_{i=1}^{k} \mathbb{1}(\hat{\mathbf{y}}_1^{(i)} \neq \hat{\mathbf{y}}_2^{(i)}) \leq \sum_{i \in S} \mathbb{1}(\hat{\mathbf{y}}_1^{(i)} \neq \hat{\mathbf{y}}_2^{(i)}) + \sum_{j \notin S} \mathbb{1}(\hat{\mathbf{y}}_1^{(j)} \neq \hat{\mathbf{y}}_2^{(j)}) \leq \Delta \cdot D$.

  We set the threshold $\tau = D \cdot d \cdot (1-\epsilon)$ in the evaluation. Point 4 guarantees that $\tau < k$ (and implicitly implies $k > D(1-\epsilon)$), so because $D \cdot \Delta \leq D \cdot d(1-\epsilon) = \tau < k$, $\mathcal{H}.\mathsf{Eval}$ will evaluate $\Delta' \leq \tau$. Thus, $\mathcal{H}.\mathsf{Eval}$ will always evaluate to $\boldsymbol{CLOSE}$ in this case, regardless of the choice of CRHF.

- Now consider $||\mathbf{x}_1 - \mathbf{x}_2||_0 \geq d(1+\epsilon)$, and again, let $\Delta = ||\mathbf{x}_1 - \mathbf{x}_2||_0$ and define $S \subset L$ and $T \subset R$ as before.

  By point 4 again ($\gamma n \leq d(1+\epsilon)$), we can restrict S to S' where $|S'| = \gamma n$, and by the properties of expanders $|N(S')| \geq \gamma n \cdot \alpha$. Now, point 3 guarantees that $\tau = D \cdot d \cdot (1-\epsilon) < \alpha \cdot \gamma n$. So, for every $i \in T'$, $\hat{\mathbf{x}}_1^{(i)} \neq \hat{\mathbf{x}}_2^{(i)}$, and $|T'| \geq \alpha \cdot \gamma n > \tau$. Now we want to argue that with all but negligible probability over our choice of g, g will preserve this equality relation, and so $\Delta' = |T'|$. Given that our expander is δ-balanced for some constant $\delta > 0$, we have that $|\hat{\mathbf{x}}_1^{(i)}| = |\hat{\mathbf{x}}_2^{(i)}| = |N(r_i)| \geq (1-\delta)nD/k$. Now, point 3 states that the constraints have $k < n/t$, implying $n/k > t$. So, $(1-\delta)D \cdot n/k > (1-\delta)D \cdot t$.

  This means that every input to g will be larger than the output ($(1-\delta)$ is a constant and $D = \omega(1)$), and so if $g(\hat{\mathbf{x}}_1^{(i)}) = g(\hat{\mathbf{x}}_2^{(i)})$ but $\hat{\mathbf{x}}_1^{(i)} \neq \hat{\mathbf{x}}_2^{(i)}$ for any i, then our adversary has found a collision, which happens with all but negligible probability for adversaries running in time $\text{poly}(\kappa)$.

  Therefore, with all but negligible probability over the choice of g and adversarially chosen $\mathbf{x}_1$ and $\mathbf{x}_2$ in this case, $\Delta' = \sum_{i=1}^{m'} \mathbb{1}(\hat{\mathbf{y}}_1^{(i)} \neq \hat{\mathbf{y}}_2^{(i)}) \geq \alpha \cdot \gamma n = \tau$, and $\mathcal{H}.\mathsf{Eval}$ outputs $\boldsymbol{FAR}$.

□

3.4.8 Proofs for Section 3.4.7: CRHFs for a Gap-Hamming PPH

Here is the proof of the lemma left out in Section 3.4.7.

Lemma 19. *For n sufficiently large, $k = O(n)$, constant $a \in (0,1)$, $\gamma = o(\frac{k}{n\log(n/k)})$, and $D = \Theta(\log(1/\gamma))$. For any constant $\delta \in (0,1)$, δ-biased (n,k,D,γ,aD)-expanders exist and are constructible in time* $\text{poly}(n)$.

Proof. First, we will show that with the right parameter settings for D and γ, we can show δ-biased (n,k,D,γ,aD)-expanders exist via random sampling. In fact, we will show that with constant probability p, we will sample such an expander. Then, we will show that with probability at least $1 - \text{negl}(n)$, the graph we sample via this method is δ-balanced. So, the probability we will sample a graph that is both an (n,k,D,γ,aD)-expander and δ-biased is a union bound: $p - \text{negl}(n)$. This means that we will sample a δ-biased (n,k,D,γ,aD)-expander with constant probability.

Sampling an expander with constant probability. First we will show that we can sample these expanders with constant probability. We will bound the following sampling procedure: for each node in L, uniformly sample D distinct neighbors in R and add those edges. We have that for any set $S \subseteq [n]$ and $K \subseteq [k]$,

$$\Pr[N(S) \subseteq K] \leq \left(\frac{|K|}{k}\right)^{D\cdot|S|}$$

since for this upper bound, we can just model this process as every node in L choosing D edges to R independently.

G is *not* a (n,k,D,γ,aD)-expander if there exists a subset S of R such that $|S| \leq \gamma n$, and a subset $K \subseteq [k]$, $|K| < aD|S|$ such that $N(S) \subseteq K$. We will upper bound this probability with a constant. We start by turning this probability into an infinite series:

$$\begin{aligned}
\Pr_G[G \text{ is not a } (\gamma,k,D,\gamma,aD)\text{-expander}] &\leq \sum_{S,|S|\leq\gamma n}\ \sum_{K,|K|=aD|S|} \Pr_G[N(S)\subseteq K] \\
&\leq \sum_{s=1}^{\gamma n}\ \sum_{S,|S|=s}\ \sum_{K,|K|=aDs} \Pr_G[N(S)\subseteq K] \\
&\leq \sum_{s=1}^{\gamma n} \binom{n}{s}\cdot\binom{k}{aDs}\cdot\left(\frac{aDs}{k}\right)^{Ds} \\
&\leq \sum_{s=1}^{\gamma n} \left(\frac{ne}{s}\right)^{s}\cdot\left(\frac{ke}{aDs}^{aDs}\right)\cdot\left(\frac{\alpha s}{k}\right)^{Ds} \\
&\leq \sum_{s=1}^{\gamma n} x'^{s} \text{ for } x' = \left(\frac{ne}{s}\right)\left(\frac{ke}{aDs}\right)^{aD}\left(\frac{aDs}{k}\right)^{D}
\end{aligned}$$

Notice that for s varying from 1 to γn, this quantity is maximized when $s = \gamma n$. So, let $x = \left(\frac{e}{\gamma}\right)\left(\frac{ke}{aD\gamma n}\right)^{aD}\left(\frac{aD\gamma n}{k}\right)^{D}$, where $x' \leq x$. We will interpret this as a geometric series and get

$$\sum_{s=1}^{\gamma n} x'^s \leq \sum_{s=1}^{\gamma n} x^s \leq \sum_{s=1}^{\infty} x^s = \frac{x}{1-x} \text{ if } |x| < 1.$$

We need this upper bound to be some constant less than 1, which means we want x to be a constant strictly less than $\frac{1}{3}$. This is where our restrictions on k, γ, and D will come into play.

$$\begin{aligned}\left(\frac{e}{\gamma}\right)\left(\frac{ke}{aD\gamma n}\right)^{aD}\left(\frac{aD\gamma n}{k}\right)^{D} &< \frac{1}{3}\\ \left(\frac{e}{\gamma}\right)e^{D}\left(\frac{aD\gamma n}{k}\right)^{D(1-a)} &< \frac{1}{3}\\ e^{D}\left(\frac{aD\gamma n}{k}\right)^{D(1-a)} &< \frac{\gamma}{3e}\end{aligned}$$

Now, we have $k = o(n)$. Assume that we have $\gamma = o(k/(n(\log(n/k))))$ and $D = O(\log(1/\gamma)))$. With these parameters, we have $D\gamma = o(k/n)$. This is fairly easy to show: if $\gamma = o(k/n(\log(n/k)))$, then $\gamma = k/(nf(\log(n/k)))$ for some $f(a) = \omega(a)$, and we have two cases to analyze:

- $\log(n/k) \geq \log(f(\log(n/k)))$. Now, because $f(a) = \omega(a)$, this means $f(\log(n/k)) = \omega(\log(n/k))$. And given the inequality, $f(\log(n/k)) = \omega(\log(n/k)+\log(f(\log(n/k)))$.
- $\log(n/k) < \log(f(\log(n/k)))$. So, $f(\log(n/k)) = \omega(\log(f(\log(n/k))))$ (and is exponentially larger). Given the inequality, this implies $f(\log(n/k)) = \omega(\log(n/k)+\log(f(\log(n/k)))$.

In both cases, we have $f(\log(n/k)) = \omega(\log(n/k) + \log(f(\log(n/k)))$. This yields

$$D\gamma = \frac{\log(n/k) + \log(f(\log(n/k)))}{f(\log(n/k))} \cdot \frac{k}{n} = o(1) \cdot \frac{k}{n} = o(k/n).$$

$D\gamma = o(k/n)$ gives us

$$\frac{aD\gamma n}{k} = \frac{a \cdot c_1 \log(1/\gamma) \cdot c_2\gamma n}{k} = \frac{o(k)}{k} = o(1).$$

Therefore, for any constant $0 < b < 1$ and sufficiently large n, we have $\frac{aD\gamma n}{k} < b$. Substituting that term with b, we have

$$\begin{aligned}e^{D}\left(\frac{aD\gamma n}{k}\right)^{D(1-a)} < e^{D} \cdot b^{D(1-a)} &< \frac{\gamma}{3e}\\ D(\log(e) + (1-a)\log(b)) &< \log(\gamma) - \log(3e)\\ D((1-a)\log(1/b) - \log(e)) &> \log(1/\gamma) + \log(3e)\\ D &> \frac{\log(1/\gamma) + \log(3e)}{(1-a)\log(1/b) - \log(e)}\end{aligned}$$

Now, since we can use any constant b, we will choose b to make the denominator a positive constant: let $b > e^{(a-1)}$, and we have that

$$D = \Omega(\log(1/\gamma)).$$

Given these matching upper and lower bounds for D, we have $D = \Theta(\log(1/\gamma))$.

Sampling a Balanced Expander. This will be a simple application of a Chernoff bound. Consider the expected value of the degree of nodes on the right: $\mathrm{E}_G[|N(v)|] = nD/k$. Notice that $|N(v)| = \sum_{u \in L} \mathbb{1}(v \in N(u))$, and for every $u \in L$ and $v \in R$, we have $\Pr_G[v \in N(u)] = D/k$, which is independent for every $u \in L$. A Chernoff bound tells us for every $v \in R$ and any constant $0 \leq d \leq 1$, $\Pr_G[|N(v)| \leq (1-\delta)nD/k] \leq \exp[\frac{-\delta^2 nD}{2k}]$. Because $k = o(n)$ and δ is constant, $\exp[\frac{-\delta^2 nD}{2k}] = 2^{-n^{O(1)}} = \mathrm{negl}(n)$. Now, via a simple union bound over all $v \in R$, we have

$$\begin{aligned}
\Pr_G[\exists v \in R, |N(v)| < (1-\delta)nD/k] &\leq \sum_{v \in R} \Pr_G[|N(v)| < (1-\delta)nD/k] \\
&\leq k \cdot \exp\left[\frac{-\delta^2 nD/k}{2}\right] \\
&= k/2^{n^{O(1)}} = \mathrm{negl}(n).
\end{aligned}$$

Therefore, the probability that our random graph is δ-balanced is at least $1 - \mathrm{negl}(n)$.

This completes the proof: we can simply sample a random D-left-regular bipartite graph, check if it is a balanced expander with the desired parameters, and with constant probability it will be. □

3.4.9 A Gap-Hamming PPH from Sparse Short Vectors

In this section, we present our second family of robust property-preserving hash (PPH) functions for gap Hamming distance. The construction proceeds in three steps: in Section 3.4.9, we start with an (unconditionally) secure non-robust PPH; in Section 3.4.9, we build on this to construct a robust PPH with a restricted input domain; and finally, in Section 3.4.9, we show how to remove the restriction on the input domain.

The construction is the same as the collision-resistant hash function construction in the work of [10]. In a single sentence, our observation is that their input-local *hash functions are* locality-sensitive *and thus give us a robust gap-Hamming PPH (albeit under a different assumption). We proceed to describe the construction in full for completeness.*

Non-Robust Gap-Hamming PPH

We first describe our starting point, a non-robust PPH for GapHamming*, derived from the locality sensitive hash of Kushilevitz, Ostrovsky, and Rabani [91]. In a nutshell, the hash function is parameterized by a random sparse $m \times n$ matrix $\mathbf{A}$ with 1s in a $1/d$ fraction of its entries and 0s elsewhere; multiplying this matrix by a vector $\mathbf{z}$ "captures" some information about the Hamming weight of $\mathbf{z}$; in particular, it distinguishes between the cases that the Hamming weight is much larger than d versus much*

Non-robust GapHamming(n, d, ϵ) PPH family $\mathcal{H}$

- $\mathcal{H}.\mathsf{Samp}(1^\kappa, 1^n)$. Pick a constant c appropriately such that $m := \frac{c\kappa}{\epsilon^2} < n$. Let

$$\mu_1 = \frac{m}{2}(1 - e^{-2(1-\epsilon)}); \quad \mu_2 = \frac{m}{2}(1 - e^{-2(1+\epsilon)}) \text{ and } \tau = (\mu_1 + \mu_2)/2$$

  Generate an $m \times n$ matrix $\mathbf{A}$ by choosing each entry from the Bernoulli distribution $\mathsf{Ber}(1/d)$. Output $(\mathbf{A}, \tau)$ as the description of the hash function.

- $\mathcal{H}.\mathsf{Hash}((\mathbf{A}, \tau), \mathbf{x})$. Output $\mathbf{Ax} \in \mathbb{Z}_2^m$.

- $\mathcal{H}.\mathsf{Eval}((\mathbf{A}, \tau), \mathbf{y}_1, \mathbf{y}_2)$. If $||\mathbf{y}_1 \oplus \mathbf{y}_2||_0 \leq \tau$, output ***CLOSE***, otherwise output ***FAR***.

Table 3.2: Construction of a non-robust GapHamming(n, d, ϵ) PPH family.

smaller. Furthermore, since this hash function is linear, it can be used to compress two inputs $\mathbf{x}$ *and* $\mathbf{y}$ *independently and later compute their Hamming* distance. *The construction is described in Table 3.2.*

Lemma 20 ([91])**.** *Let* κ *be a security parameter. For every* $n \in \mathbb{N}$, $d \in [n]$ *and* $\epsilon = \Omega(\sqrt{\kappa/n})$*, Construction 3.2 is a non-robust PPH for* GapHamming(n, d, ϵ).

Proof. First, we note that $\mathcal{H}$ is compressing. We have $m = \frac{c\kappa}{\epsilon^2}$ and $\epsilon = \Omega(\sqrt{\kappa/n})$. Therefore, if we have chose c appropriately, there exists another constant $c' < 1$ such that $m \leq c'n$. Compression is why we require a lower bound on epsilon.

Now, we show that $\mathcal{H}$ satisfies the non-robust notion of correctness. For any $\mathbf{x}_1$ and $\mathbf{x}_2 \in \{0,1\}^n$, let $\mathbf{z} = \mathbf{x}_1 \oplus \mathbf{x}_2$. $\mathcal{H}.\mathsf{Eval}(\mathbf{Ax}_1, \mathbf{Ax}_2)$ tests if $||\mathbf{Az}||_0 \leq \tau$. We will show that for all $\mathbf{z}$, with all but negligible probability over our choice of $\mathbf{A}$, this threshold test will evaluate correctly.

To do this, we will invoke the (information theoretic) XOR lemma. That is, if $||\mathbf{z}||_0 = k$, then for $\mathbf{a}_i \xleftarrow{\$} \mathsf{Ber}(p)^n$, $\Pr[\mathbf{a}_i \cdot \mathbf{z} = 1] = \frac{1}{2}(1 - (1-2p)^k)$. Our hash function has $p = 1/d$, so now, we will apply this to our two cases for $\mathbf{z}$:

- $||\mathbf{z}||_0 \leq d(1-\epsilon)$. We have that $\Pr[\mathbf{a}_i \cdot \mathbf{z} = 1] = \frac{1}{2}(1 - (1 - \frac{2}{d})^{d(1-\epsilon)}) \sim \frac{1}{2}(1 - \frac{1}{e^{2(1-\epsilon)}})$ by the XOR lemma. We get that for all $\mathbf{z}$ such that $||\mathbf{z}||_0 \leq d(1-\epsilon)$,

$$\mathrm{E}_{\mathbf{A}}[||\mathbf{Az}||_0] = m \cdot \Pr_{\mathbf{a}_i}[\mathbf{a}_i \cdot \mathbf{z} = 1] \leq \frac{m}{2}(1 - \frac{1}{e^{2(1-\epsilon)}}) := \mu_1$$

- $||\mathbf{z}||_0 \geq d(1+\epsilon)$. Again, using the XOR lemma, and plugging in $k = d(1+\epsilon)$, we have that,

$$\mathrm{E}_{\mathbf{A}}[||\mathbf{Az}||_0] = m \cdot \Pr_{\mathbf{a}_i}[\mathbf{a}_i \cdot \mathbf{z} = 1] \geq \frac{m}{2}(1 - \frac{1}{e^{2(1+\epsilon)}}) := \mu_2$$

Now, a routine calculation using Chernoff bounds shows that with overwhelming probability over our choice of $\mathbf{A}$, we do not miscategorize the Hamming weight of $\mathbf{z}$. We will go through both cases again.

- $||\mathbf{z}||_0 \leq d(1-\epsilon)$. Recall that $\mu_1 = \frac{m}{2}(1 - \frac{1}{e^{2(1-\epsilon)}})$. Let $\tau = (1+\delta)\mu_1$. A Chernoff bound states that

$$\Pr_{\mathbf{A}}[||\mathbf{Az}||_0 \geq \tau] \leq e^{-\delta\mu_1/3}$$

  Now we will ensure that $e^{-\delta\mu_1/3} = \text{negl}(\kappa)$. Using τ, we can compute δ exactly to be $\frac{1}{2}(\frac{e^{-2(1-\epsilon)} - e^{-2(1+\epsilon)}}{(1-e^{-2(1-\epsilon)})})$. We will use the fact that $e^{2\epsilon} - e^{-2\epsilon} > 4\epsilon$ for all $\epsilon > 0$. Now, using our expression for μ_1, we get that

$$\begin{aligned}
e^{-\delta\mu_1/3} &= \exp[-\delta \cdot \frac{m}{2}(1 - e^{-2(1-\epsilon)})] \\
&= \exp[-\frac{1}{2} \cdot \left(\frac{e^{-2(1-\epsilon)} - e^{-2(1+\epsilon)}}{1 - e^{-2(1-\epsilon)}}\right) \cdot \frac{c\kappa}{2\epsilon^2}(1 - e^{-2(1-\epsilon)})] \\
&= \exp[-\frac{1}{2}(e^{-2(1-\epsilon)} - e^{-2(1+\epsilon)}) \cdot \frac{c\kappa}{2\epsilon^2}] \\
&\sim \exp[-\frac{1}{2} \cdot \frac{4\epsilon}{e^2} \cdot \frac{c\kappa}{2\epsilon^2}] = \exp[\frac{-c\kappa}{e^2\epsilon}] = \text{negl}(\kappa)
\end{aligned}$$

- $||\mathbf{z}||_0 \geq d(1+\epsilon)$. Recall that $\mu_2 = \frac{m}{2}(1 - \frac{1}{e^{2(1+\epsilon)}})$. Given our expression of τ, we have that $\tau = (1-\delta)\mu_2$. We will use the same strategy as in the previous case to show correctness, showing

$$e^{-\delta^2\mu_2/2} \leq e^{-\kappa}$$

  So, recall that $\delta = \frac{1}{2}(\frac{e^{-2(1-\epsilon)} - e^{-2(1+\epsilon)}}{(1-e^{-2(1-\epsilon)})})$, and notice that $(1 - e^{-2(1+\epsilon)}) \geq 1 - e^{-2} > \frac{4}{5}$ for all $\epsilon > 0$. We compute

$$\begin{aligned}
e^{-\delta^2\mu_2/2} &= \exp[-\delta^2 \frac{m}{2}(1 - e^{-2(1+\epsilon)})] \\
&\leq \exp[-\delta^2 \frac{c\kappa}{2\epsilon^2} \cdot \frac{4}{5}] = \exp[-\frac{2}{5} \cdot \delta^2 \frac{c\kappa}{\epsilon^2}] \\
&\leq \exp[-\frac{2}{5} \cdot \frac{1}{4}\left(\frac{e^{-2(1-\epsilon)} - e^{-2(1+\epsilon)}}{(1 - e^{-2(1-\epsilon)})}\right)^2 \cdot \frac{c\kappa}{\epsilon^2}] \\
&\leq \exp[-\frac{1}{10} \cdot \left(\frac{e^{-2(1-\epsilon)} - e^{-2(1+\epsilon)}}{1}\right)^2 \cdot \frac{c\kappa}{\epsilon^2}] \\
&\leq \exp[-\frac{1}{10} \cdot (e^{-2}(e^{2\epsilon} - e^{-2\epsilon}))^2 \cdot \frac{c\kappa}{\epsilon^2}] \\
&\leq \exp[-\frac{1}{10} \cdot \frac{16\epsilon^2}{e^4} \cdot \frac{c\kappa}{\epsilon^2}] = \exp[-\frac{8c\kappa}{5e^4}] = \text{negl}(\kappa)
\end{aligned}$$

In both cases, the probability that we choose a bad matrix $\mathbf{A}$ for a fixed input $\mathbf{z}$ is negligible in the security parameter, proving the lemma. □

Robust Gap-Hamming PPH with a Sparse Domain

To make the construction robust, we need to protect against two directions of attack: finding a low-weight vector that gets mapped to a high-weight vector, and finding a high-weight vector that maps to a low-weight one. To address the first line of attack, we will use an information-theoretic argument identical to the one in the proof of lemma 20. In short, in the proof of lemma 20, we computed the probability that a fixed *low-weight vector maps to a high-weight vector (on multiplication by the sparse matrix* $\mathbf{A}$*). The number of low-weight vectors is small enough that by a union bound, the probability that* there exists *a low-weight vector that maps to a high-weight vector is small as well.*

To address the second line of attack, we unfortunately cannot make an information-theoretic argument (to be expected, as we compress beyond the information theoretic limits). Indeed, one possible attack is simply to use Gaussian elimination on $\mathbf{A}$ *to come up with non-zero (probably high-weight) vector that maps to 0. Because* $\mathbf{A}$ *has a non-trivial null-space, this attack is likely to succeed.*

To thwart such attacks, we leverage one of the following two loopholes that circumvent Gaussian elimination: (1) our first approach is to consider linear functions with sparse *domains, restricting the input to be vectors of weight* $\leq \beta n$ *for constant* $\beta < 1/2$*, and so Gaussian elimination no longer works; and (2) building on this, we extend this to a* non-linear *construction where the domain is the set of all strings of a certain length. Our construction relies on the following hardness assumption that we refer to as the "sparse short vector" (SSV) assumption. The SSV assumption is a variant of the syndrome decoding problem on low-density parity-check codes, and roughly states that it is hard to find a preimage of a low-weight syndrome vector for which the preimage has "medium" weight.*

Definition 21. *Let* $n \in \mathbb{N}$*. Let* $\hat{\beta}, \alpha, \eta \in [0,1]$ *and* $\omega, \tau \in [n]$*. The* $(\hat{\beta}, \alpha, \omega, \tau, \eta)$*-SSV (Sparse Short Vector) assumption states that for any PPT adversary* $\mathcal{A}$ *given an* $\eta n \times n$ *matrix* $\mathbf{A}$ *with entries sampled from* $\mathsf{Ber}(\alpha)$*,*

$$\Pr_{\mathbf{A} \sim \mathsf{Ber}(\alpha)^{\eta n \times n}} \left[\mathcal{A}(\mathbf{A}) \to \mathbf{z} \in \{0,1\}^n : \omega \leq ||\mathbf{z}||_0 \leq \hat{\beta} n \text{ and } ||\mathbf{Az}||_0 \leq \tau \right] = \text{negl}(n).$$

On the Assumption. *We now consider attacks on the SSV assumption which help us refine the parameter settings.*

One way to attack the assumption is to solve the syndrome decoding problem for sparse parity-check matrices (also called the binary short vector problem or bSVP in [10]). In particular, find a $\hat{\beta}$*-sparse* $\mathbf{z}$ *such that* $\mathbf{Az} = 0$*. To thwart these attacks, and at the same time have a compressing PPH construction, we need at the very minimum that* $H(\hat{\beta}/2) > \eta > 2\hat{\beta}$*.*

$\eta < H(\hat{\beta}/2)$ *ensures compression. Recall that we hash elements* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *in the hopes of being able to approximate their hamming distance. We have* $\mathbf{z} = \mathbf{x}_1 \oplus \mathbf{x}_2$ *is the vector we want to compute gap-hamming on, and so to guarantee* $\mathbf{z}$ *has sparsity* $\hat{\beta}$*,* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *need sparsity* $\hat{\beta}/2 = \beta$*. Vector* $\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ *of weight at most* $\hat{\beta} n/2$ *require (asymptotically)* $H(\hat{\beta}/2)n$ *bits each to describe, and so* ηn *needs to be less than that.*

$\eta > 2\hat{\beta}$ is for security. If $\eta n \leq 2\hat{\beta}n$, then we are able to use Gaussian elimination to find a $\hat{\beta}$-sparse vector. Consider an $\eta n \times n$ matrix $\mathbf{A}$, and the first $\eta n \times \eta n$ square of it, call it $\mathbf{A}'$. Use Gaussian elimination to compute an ηn-length vector $\mathbf{z}'$ such that $\mathbf{A}'\mathbf{z}' = \mathbf{0}$. Padding $\mathbf{z}'$ with 0's, we get $\mathbf{z}$ where $\mathbf{Az} = \mathbf{0}$. We expect $||\mathbf{z}||_0 = \eta n/2$, and since $\eta n/2 \leq \hat{\beta}n$, we have broken the assumption.

Thus, for $\beta = \hat{\beta}/2$, we would like η to be as close to $H(\beta)$ as possible to give us non-trivial compression and at the same time, security from as conservative an assumption as possible. The reader might wonder about efficient unique decoding algorithms for LDPC codes. It turns out that the noise level ($\hat{\beta}$) for which the efficient decoding algorithms for LDPC imply a solution to bSVP is only a subset of the entire range $(H^{-1}(\eta), \eta/2)$. The range where efficient algorithms do not work (the "gap") grows with the locality parameter α [62, 55], and as the sparsity $\hat{\beta}$ tends towards 0, LDPC becomes similar to random linear code both combinatorially [61, 100], and, presumably, computationally. For a more detailed discussion, we refer the reader to [10].

We will set the sparsity parameter $\alpha \geq c/n$ for a large enough constant c, or to be conservative $\alpha \geq \log n/n$ to ensure that w.h.p. there are no all-0 columns.

Finally, we point out that when the matrix $\mathbf{A}$ is uniformly random and not sparse, SSV (where the adversary has to map small vectors to tiny vectors) is equivalent to bSVP (where the adversary has to map small vectors to 0). We briefly sketch how to reduce bSVP to SSV for a uniformly random $\mathbf{A}$. The reduction takes an instance of bSVP, a matrix $\mathbf{B} = [\mathbf{B}_1||\mathbf{B}_2]$ where $\mathbf{B}_2$ is square, and generates the matrix $\mathbf{A} := \mathbf{B}_2^{-1}\mathbf{B}_1$ as an instance of SSV. If the adversary finds $\mathbf{x}_1, \mathbf{x}_2$ such that $\mathbf{B}_2^{-1}\mathbf{B}_1\mathbf{x}_1 = \mathbf{x}_2$ solving SSV, then we have $[\mathbf{B}_1||\mathbf{B}_2] \cdot [\mathbf{x}_1^T||\mathbf{x}_2^T] = 0$, solving bSVP. However, this reduction does not work when $\mathbf{A}$ is sparse, though this connection indicates that the SSV problem is also hard.

Robust Hashing Construction for Gap-Hamming. *Let the problem β-*Sparse Gap-Hamming *be the same as* GapHamming*, except we restrict the domain to be over $\mathbf{x} \in \{0,1\}^n$ with sparsity $||\mathbf{x}||_0 \leq \beta n$. Our construction of a robust PPH for* Sparse Gap-Hamming *is as follows.*

Settings Parameters from SSV Assumption to the Sparse Domain Construction *Our main tool in this construction is the SSV assumption. So, here we consider a very conservative parameter setting for the SSV assumption, and show what parameters we acheive for our β-Sparse Gap-Hamming construction.*

- *$n \in \mathbb{N}$ and $\epsilon = \Omega(\sqrt{\kappa/n})$.*
- *Let the $(\beta, \alpha, \omega = (1+\epsilon)/\alpha, \tau, \eta)$-SSV be true for the following parameters: $0 < \beta \leq 0.04$ is a constant (this needs to be true in order to have $4\beta < H(\beta)$), $\alpha \geq \log n/n$, $\tau = \frac{\eta n}{4}(e - e^{-2(1-\epsilon)} - e^{-2(1+\epsilon)})$, and $\eta = H(\beta) \cdot (1-\zeta)$ for a small constant ζ. These parameters come from believing a relatively conservative parameter settings for the SSV.*
- *Notice that the η in the assumption is just the number of output bits over the number of input bits. The actual compression of our construction is actually*

Robust PPH for Sparse GapHamming(n, d, ϵ)

- $\mathcal{H}.\mathsf{Samp}(1^\kappa, n, d, \beta, \epsilon)$:
  - Choose appropriate constants $c_1, c_2 > 0$ such that

$$m := \max\left\{\frac{c_1\kappa}{\epsilon^2}, \frac{n \cdot 3e^2 \ln(2) H(d(1-\epsilon)/n) + c_2\kappa}{\epsilon}, 4\beta n + 1\right\} < H(\beta)n\,.$$

  - Compute $\mu_1 = \frac{m}{2}(1 - e^{-2(1-\epsilon)})$ and $\mu_2 = \frac{m}{2}(1 - e^{-2(1+\epsilon)})$. Let $\tau = (\mu_1 + \mu_2)/2$.
  - Generate an $m \times n$ matrix $\mathbf{A}$ by choosing each entry from $\mathsf{Ber}(1/d)$.
  - Output $\mathbf{A}$ and the threshold τ.
- $\mathcal{H}.\mathsf{Hash}(\mathbf{A}, \mathbf{x})$: If $||\mathbf{x}||_0 \leq \beta n$, output $\mathbf{Ax} \in \mathbb{Z}_2^m$. Otherwise, output failure.
- $\mathcal{H}.\mathsf{Eval}(\mathbf{y}_1, \mathbf{y}_2)$: if $||\mathbf{y}_1 \oplus \mathbf{y}_2||_0 \leq \tau$, output ***CLOSE***, otherwise output ***FAR***.

Table 3.3: Construction of a robust PPH for sparse-domain GapHamming(n, d, ϵ).

(at most) the number of output bits over $H(\beta)n$. So, with this assumption, we get compression $(\eta H(\beta)n)/(H(\beta)n) = (1 - \zeta) = \Omega(1)$, and a center for our Gap-Hamming problem to be at any $d \leq \frac{n}{\log n}$.

We formally show why assuming the SSVunder the correct parameter settings yields a β-Sparse Gap-Hamming PPH.

Theorem 18. *Let κ be a security parameter, let $n = \mathsf{poly}(\kappa) \in \mathbb{N}$ and take any $\epsilon = \Omega(\sqrt{\kappa/n})$. Let $\beta, \alpha, \eta \in [0, 1]$ and $\omega, \tau \in [n]$. Under the $(\beta, \alpha, \omega = (1+\epsilon)/\alpha, \tau, \eta)$-SSV assumption, the construction in Table 3.3 is a direct-access secure PPH for $\beta/2$-sparse* GapHamming(n, d, ϵ) *where $d = 1/\alpha$.*

Proof. This proof follows the same structure as the proof for lemma 20, with the same computations for μ_1 and μ_2, but we will require a different m to get an information-theoretic argument for the case when $||\mathbf{z}||_0 \leq d(1-\epsilon)$ and rely on the assumption to show robustness for the other case.

So, let $\mu_1 = \frac{m}{2}(1 - \frac{1}{e^{2(1-\epsilon)}})$ and $\mu_2 = \frac{m}{2}(1 - \frac{1}{e^{2(1+\epsilon)}})$. δ is also computed as before to be $\delta = \frac{1}{2}(\frac{e^{-2(1-\epsilon)} - e^{-2(1+\epsilon)}}{(1-e^{-2(1-\epsilon)})})$. We analyze both cases with two claims.

Claim 11. *Given any adversary $\mathcal{A}$, it is impossible for $\mathcal{A}$ to output a vector $\mathbf{z}$ such that $||\mathbf{z}||_0 < d(1-\epsilon)$ and $||\mathbf{Az}||_0 \geq \tau$ with all but negligible probability over our choice of $\mathbf{A}$.*

Proof. We get, via a union bound and then Chernoff bound. Recall that $m = \max\left\{\frac{c_1\kappa}{\epsilon^2}, \ \frac{1}{\epsilon}\left(n \cdot 3e^2 \ln(2) H(d(1-\epsilon)/n) + c_2\kappa\right)\right\}$, and so $m \geq \frac{n \cdot 3e^2 \ln(2) H(d(1-\epsilon)/n) + c_2\kappa}{\epsilon}$.

Also, notice that $\delta \geq \frac{2\epsilon}{e^2(1-e^{-2(1-\epsilon)})}$.

$$\begin{aligned}
&\Pr_{\mathbf{A}}[\exists \mathbf{z} \text{ s.t. } ||\mathbf{z}||_0 \leq d(1-\epsilon) \wedge ||\mathbf{Az}||_0 \geq \tau] \leq 2^{nH(d(1-\epsilon)/n)} \Pr_{\mathbf{A}}[||\mathbf{Az}||_0 \geq \tau] \\
&\quad \leq \exp[\ln(2) nH(d(1-\epsilon)/n) - \delta\mu_1/3] \\
&\quad \leq \exp[\ln(2) H(d(1-\epsilon)/n)n - \frac{2\epsilon}{e^2(1-e^{-2(1-\epsilon)})} \cdot \frac{m}{2}\left(1-e^{-2(1-\epsilon)}\right) \cdot \frac{1}{3}] \\
&\quad = \exp[\ln(2) H(d(1-\epsilon)/n)n - \frac{\epsilon}{3e^2} \cdot m] \\
&\quad \leq \exp[\ln(2) H(d(1-\epsilon)/n)n - \frac{\epsilon}{3e^2} \cdot \left(\frac{3e^2}{\epsilon} \cdot (\ln(2) H(d(1-\epsilon)/n)n) + \frac{c_2\kappa}{\epsilon}\right)] \\
&\quad = \exp[-\frac{\epsilon}{3e^2} \cdot \frac{c_2\kappa}{\epsilon}] = \exp[-\frac{c_2}{3e^2}] = \text{negl}(\kappa)
\end{aligned}$$

□

Claim 12. *Let $\eta = m/n$, β be the parameter input into $\mathcal{H}.\mathsf{Samp}$, and τ the threshold computed in $\mathcal{H}.\mathsf{Samp}$. Assuming the $(2\beta, 1/d, d(1+\epsilon), \tau, \eta)$-SSV assumption, any PPT adversary $\mathcal{A}$ cannot find $\mathbf{z}$ such that $||\mathbf{z}||_0 \geq d(1+\epsilon)$ and $||\mathbf{Az}||_0 \leq \tau$.*

Proof. We need to use the assumption to bound the probability an adversary $\mathcal{A}$ is able to produce two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ in $\{0,1\}^n$ such that $||\mathbf{x}_1||_0, ||\mathbf{x}_2||_0 \leq \beta n$, $||\mathbf{x}_1 - \mathbf{x}_2|| \geq d(1+\epsilon)$, and $||\mathbf{Ax}_1 \oplus \mathbf{Ax}_2||_0 \leq \tau$, when given $\mathbf{A}$. That is, equivalently, it can produce a vector $\mathbf{z} \in \{0,1\}^n$ where $d(1+\epsilon) \leq ||\mathbf{z}||_0 \leq 2\beta n$ and $||\mathbf{Az}||_0 \leq \tau$. So, the statement becomes exactly the definition of the $(2\beta, 1/d, d(1+\epsilon), \tau, \eta)$-SSV assumption:

$$\Pr_{\mathbf{A}}[\mathcal{A}(\mathbf{A}) \to \mathbf{z} \in \{0,1\}^n : \tau \leq ||\mathbf{z}||_0 \leq 2\beta n \wedge ||\mathbf{Az}||_0 \leq \tau] = \text{negl}(n)$$

□

The claims work together to show that no PPT adversary can find low weight vectors that map to high weight ones, and vice-versa, even if she has access to the code of hash function, $\mathbf{A}$. Therefore, the construction is robust in the direct-access model. □

From the Full Domain to a Sparse Domain

Now that we have a gap-Hamming preserving hash for sparse vectors ($||\mathbf{x}||_0 \leq \beta n$), we can extend this to work for the full domain.

One might consider a trivial way of converting any vector into a sparse vector via padding with 0's; we can take any vector $\mathbf{x} \in \{0,1\}^n$ and convert it into a 'sparse' vector $\mathbf{x}' \in \{0,1\}^{n'}$ with density at most n/n' by padding it with $n'-n$ zeros. However, this transformation is expensive in the length of the vector; we need to more than quadruple the length of $\mathbf{x}$ to get the density to be $\beta < .04$. Unfortunately, this transformation is also linear*, and if we believe that the non-sparse version (construction 3.2) is not robust, then this combined linear version cannot be robust with the same parameters.*

Robust GapHamming(n, d, ϵ) PPH family $\mathcal{H}$

- $\mathcal{H}.\mathsf{Samp}(1^\kappa, n, d, \beta, \epsilon)$.
  - If $\frac{1-1/\log(1/\beta)}{1+1/\log(1/\beta)} \geq \epsilon$, output failure.
  - Let $n' = \frac{n}{\log(1/\beta)\beta}$, $d' = \left((1-\epsilon)d + (1+\epsilon)\frac{d}{\log(1/\beta)}\right)$, and $\epsilon' = 1 - \frac{(1-\epsilon)d}{d'}$.
  - Pick constants c_1, c_2 such that

$$m := \max\left\{\frac{c_1\kappa}{\epsilon'^2}, \frac{n' \cdot 3e^2\ln(2)H(d'(1-\epsilon')/n') + c_2\kappa}{\epsilon'}, 4\beta n' + 1\right\} < n$$

  - Compute $\mu_1 = \frac{m}{2}(1 - e^{-2(1-\epsilon')})$ and $\mu_2 = \frac{m}{2}(1 - e^{-2(1+\epsilon')})$. Let $\tau = (\mu_1 + \mu_2)/2$.
  - Generate an $m \times n'$ matrix $\mathbf{A}$ by choosing each entry from $\mathsf{Ber}(1/d')$.
  - Output $\mathbf{A}$ and the threshold τ.
- $\mathcal{H}.\mathsf{Hash}(\mathbf{A}, \mathbf{x})$: let $x' \leftarrow \mathsf{Sparsify}(x', \log(1/\beta))$, output $\mathbf{A}\mathbf{x}' \in \mathbb{Z}_2^m$.
- $\mathcal{H}.\mathsf{Eval}(\mathbf{y}_1, \mathbf{y}_2)$: if $||\mathbf{y}_1 - \mathbf{y}_2||_0 \leq \tau$, output ***CLOSE***, otherwise output ***FAR***.

Table 3.4: Construction of a robust GapHamming(n, d, ϵ) PPH family.

Instead of using padding, we will use a non-linear transformation that is more efficient in sparsifying a vector in terms of length, but incurs some bounded error in measuring gap-hamming distance. As long as we are liberal enough with the gap, ϵ, this will be a robust PPH that gets around attacks on a linear sketches, despite incurring some error.

Sparsify Algorithm

Algorithm 3.4.9 takes a dense bit vector of length n and turns it into a $1/2^k$-sparse bit vector of length $2^k n/k$. This is done by breaking the vector $x \in \{0,1\}^n$ into n/k blocks of k bits, and replacing each k-bit value with its corresponding (unit) indicator vector in $\{0,1\}^{2^k}$. Given two vectors $\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ with $||\mathbf{x}_1 - \mathbf{x}_2||_0 = \Delta$, the sparsified versions $\mathbf{x}'_1$ and $\mathbf{x}'_2$ have $2\Delta/k \leq ||\mathbf{x}_1 - \mathbf{x}_2|| \leq 2\Delta$.

Recall that the trivial sparsifying method, simple padding, goes from n bits to n/β. If we let $k = \log(1/\beta)$, then using Algorithm 3.4.9, we go from n bits to $\frac{n2^k}{k} = \frac{n}{\log(1/\beta)\beta}$, saving a log factor of $1/\beta$. The construction in Table 3.4 is for dense gap-hamming.

Parameter settings for the full-domain construction *Just as in the sparse case, we will propose a parameter setting compatible with a conservative instantiation of the SSV assumption.*

- *Let $n \in \mathbb{N}$, $\beta < 0.01$, and $\epsilon' \geq \frac{1}{\log(1/\beta)+1} \approx 0.13$.*

- *We will be using the same parameter setting as for the sparse case (notice that ϵ' is larger than in the sparse setting). So, let the $(\beta, \alpha, \omega = (1+\epsilon')/\alpha, \tau, \eta')$-SSV be true for the following parameters from the sparse case: $\beta \leq 0.01$, $\alpha \geq \log n/n$, and $\tau = \frac{\eta' n}{4}(e - e^{-2(1-\epsilon')} - e^{-2(1+\epsilon')})$. Now, we will need a better compression term than before. Let $z > 0$ be a constant (close to 0), and $\eta' = \beta \log(1/\beta)(1-z) \approx 0.066(1-z)$. These parameters come from believing a relatively conservative parameter settings for the SSV assumption, with the parameters tuned just right to imply a Gap-Hamming PPH for the full domain.*

- *In total, Construction 3.4 is an η-compressing* GapHamming(n, d, ϵ) *PPH, where $\eta = \frac{\eta'}{\beta \log(1/\beta)} = (1-z)$, $d \leq \frac{1}{2\alpha}\left((1-\epsilon') + \log(1/\beta)\epsilon'\right) \approx \frac{0.87n}{\log n}$, and gap $\epsilon \geq \frac{1 - 1/\log(1/\beta)}{1 + 1/\log(1/\beta)} \approx 0.74$.*

  These parameters are formally proved to hold due to the security of the sparse construction, Construction 3.3, in Lemma 21.

Next, we will go into the details for why certain settings of the SSV assumption imply a Gap-Hamming PPH.

Theorem 19. *Let κ be a security parameter, let $n = \mathsf{poly}(\kappa) \in \mathbb{N}$. Let $\beta, \eta \in [0,1]$ and $\tau \in [n]$.*

Assuming the $(2\beta, 1/d', d'(1+\epsilon'), \tau, \eta)$-SSV assumption, where β is sparsity, $\eta = m/n'$, and τ is computed as in Table 3.4, then the construction in Table 3.4 is a Direct-Access secure PPH.

Proof. This is a simple application of theorem 18 with the correctness of algorithm Sparsify.

First, some properties about Sparsify. When given the input of an n-bit vector $\mathbf{x}$, and parameter $\log(1/\beta)$, the inner loop executes $n/\log(1/\beta)$ times. Each loop adds $2^{\log(1/\beta)} = 1/\beta$ coordinates to $\mathbf{x}'$, and so the output vector $\mathbf{x}'$ is $\frac{n}{\log(1/\beta)\beta}$ bits. Second, $\mathbf{x}'$ is β-sparse. This is because each loop adds at most one coordinate with a 1 in it (a standard basis vector), and the loop executes $n/\log(1/\beta)$ times, meaning the density is at most $\frac{n/\log(1/\beta)}{n/\log(1/\beta)\cdot 1/\beta} = \beta$.

In order for this to be a PPH, we first need compression: sparsification expands inputs and so we need to be sure that we shrink it enough afterwards. Of course the construction fails any time $m \geq n$, but we need to argue such an m even exists. As per earlier analysis, we need $m > 4\beta n'$. Given $n' = n/(\log(1/\beta)\beta)$, this means $m > 4n/\log(1/\beta)$. If $\log(1/\beta) \geq 5$, then there exists an m such that $4n/\log(1/\beta) < m < n$, and so there exists a compressing m in this context for sufficiently large n.

Now note that when given $\mathbf{x}_1$, and $\mathbf{x}_2$ where $||\mathbf{x}_1 - \mathbf{x}_2||_0 = \Delta$, we have that $\frac{2\Delta}{\log(1/\beta)} \leq ||\mathsf{Sparsify}(\mathbf{x}_1) - \mathsf{Sparsify}(\mathbf{x}_2)||_0 \leq 2\Delta$; so Sparsify introduces some error.

So, assume $(2\beta, 1/d', d'(1+\epsilon'), \tau, \eta)$-SSV holds for $\tau, \beta, \eta, d', \epsilon'$ computed as in the construction and for a contradiction assume there exists an adversary $\mathcal{A}$ that can break Direct-Access robustness of construction 3.4. We will show that $\mathcal{A}$ must break the $(2\beta, 1/d', d(1+\epsilon), \tau, \eta)$-SSV assumption. So, $\mathcal{A}$ will output two vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$ that with noticeable probability will fit into one of two cases breaking Direct-Access robustness:

- $||\mathbf{x}_1 \oplus \mathbf{x}_2||_0 < d(1-\epsilon)$ such that $||\mathbf{A}(\mathsf{Sparsify}(\mathbf{x}_1) - \mathsf{Sparsify}(\mathbf{x}_2))||_0 > \tau$. Let $\hat{\mathbf{z}} \leftarrow \mathsf{Sparsify}(\mathbf{x}_1) - \mathsf{Sparsify}(\mathbf{x}_2)$. We have that $||\hat{\mathbf{z}}||_0 < d(1-\epsilon) \leq 2d'(1-\epsilon')$. Now because of how we computed m, the same proof as in the proof of construction 3.3 will show that there exists such a $\hat{\mathbf{z}}$ with negligible probability in κ. Therefore, with all-but-negligible probability, $\mathcal{A}$ cannot take this line of attack.

- $||\mathbf{x}_1 \oplus \mathbf{x}_2||_0 > d(1+\epsilon)$ such that $||\mathbf{A}(\mathsf{Sparsify}(\mathbf{x}_1) - \mathsf{Sparsify}(\mathbf{x}_2))||_0 < \tau$. Let $\hat{\mathbf{z}} \leftarrow \mathsf{Sparsify}(\mathbf{x}_1) - \mathsf{Sparsify}(\mathbf{x}_2)$. Recall that $\mathsf{Sparsify}$ introduces bounded error, so we know that $||\hat{\mathbf{z}}||_0 > \frac{2d(1+\epsilon)}{\log(1/\beta)} \geq d'(1+\epsilon')$ given how we have computed our parameters.

  Therefore, we have computed a 2β-sparse vector $\hat{\mathbf{z}} \in \{0,1\}^{n'}$, with τ computed as in construction 3.3 for parameters n', d', ϵ', which exactly violates the $(2\beta, 1/d', d'(1+\epsilon'), \tau, \eta)$-SSV.

□

On Feasibility Settings for the Full-Domain Construction. *Here we will prove that if there exists a parameter setting for the sparse construction, Construction 3.3, with good-enough compression, then there exists a parameter setting for the full domain, Construction 3.4. It is important to note, however, that we get a worse lower bound for our resulting gap ϵ: η' is constant, so β is also constant, and since we require $\epsilon' > 1/(\log(1/\beta)+1)$ our resulting ϵ is also constant.*

Lemma 21. *Assume that Construction 3.3 is an η'-compressing robust PPH for* $\textsc{GapHamming}(n', d', \epsilon')$ *where $\eta' < \beta\log(1/\beta)$ and $\epsilon' > \frac{1}{\log(1/\beta)+1}$. Then, Construction 3.4 is an η-compressing robust PPH for* $\textsc{GapHamming}(n, d, \epsilon)$ *for the following parameters:*

- $\eta = \frac{\eta'}{\beta\log(1/\beta)}$,
- $n = \beta\log(1/\beta)n'$,
- $d = \frac{d'}{2}\left((1-\epsilon') + \log(1/\beta)\epsilon'\right)$,
- $\epsilon = \frac{\epsilon'\log(1/\beta)-(1-\epsilon')}{\epsilon'\log(1/\beta)+(1-\epsilon')}$.

Proof. Note that we can allways assume the gap is bigger, so if $\epsilon' \leq \frac{1}{\log(1/\beta)+1}$, then we can use our PPH for $\textsc{GapHamming}(n', d', \epsilon')$ as a PPH for $\textsc{GapHamming}(n', d', \frac{1}{\log(1/\beta)+1} + .0001)$. So, without loss of generality, assume $\epsilon' > \frac{1}{\log(1/\beta)+1}$.

We will show that Construction 3.4 is a robust PPH for $\textsc{GapHamming}(n, d, \epsilon)$ by showing that we can take any dense input in $\{0,1\}^n$, turn it into a sparse input in $\{0,1\}^{n'}$, and then using Construction 3.3, that hash functions and evaluations will produce correct results for $\textsc{GapHamming}(n, d, \epsilon)$. Robustness follows from the robustness of Construction 3.3.

First, compression is guaranteed since $\eta n = \eta' n' < \beta \log(1/\beta) n' = n$, implying $\eta < 1$.

Next, take any $\mathbf{x} \in \{0,1\}^n$ and let $\mathbf{x}' = \mathsf{Sparsify}(\mathbf{x}, \log(1/\beta))$. Notice that $\mathbf{x}'$ has length $n' = n/(\beta \log(1/\beta))$ and sparsity at least β by the correctness of $\mathsf{Sparsify}$. Therefore, $\mathbf{x}$ is a valid input to the hash functions from Construction 3.3.

Now, we need to show that the resulting hash function is correct. For any $\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$, where $||\mathbf{x}_1 - \mathbf{x}_2|| = \Delta$, we have $\frac{2\Delta}{\log(1/\beta)} \leq ||\mathsf{Sparsify}(\mathbf{x}_1) - \mathsf{Sparsify}(\mathbf{x}_2)|| \leq 2\Delta$. We have two cases to consider to ensure correctness.

- If $\Delta < d(1-\epsilon)$, then $||\mathbf{x}_1 - \mathbf{x}_2|| < 2d(1-\epsilon) = d'(1-\epsilon')$. Then, by the robustness of Construction 3.3, the hash function will output ***CLOSE*** with all but negligible probability over our choice of hash, even with adversarially chosen inputs.

- If $\Delta > d(1+\epsilon)$, then $||\mathbf{x}_1 - \mathbf{x}_2|| > 2d(1+\epsilon)/\log(1/\beta) = d'(1+\epsilon')$. Then, by the robustness of Construction 3.3, the hash function will output ***FAR*** with all but negligible probability over our choice of hash, even with adversarially chosen inputs.

□

Notice that setting n,d, and ϵ to these values exactly translates into having n', d', and ϵ' be the intermediate values in Construction 3.4. With this lemma we can explicitly characterize the valid parameter settings for the full domain as follows.

Full-Domain Parameter Settings. *Fix our input size n, and assume that the Sparse-Domain construction works for any constant compression η', any $\epsilon = \Omega(1)$, for some constant sparsity β.*

- *We can compress by any constant $\eta = O(1)$,*
- *we can handle any constant gap $\epsilon = \Omega(1)$,*
- *and we can let our center be any $d \leq \frac{n}{2\log n}((1-\epsilon) + (1+\epsilon))$.*

3.5 Necessity of Cryptographic Assumptions

Recall the goal of robust PPH is to compress beyond the information theoretic limits, to a regime where incorrect hash outputs exist but are computationally hard to find. When the hash function is given, this inherently means such constructions necessitate cryptographic hardness assumptions. A natural question is what types of assumptions are required to build non-trivial PPHs of various kinds.

In this section, we address the computational assumptions implied by PPH for classes of predicates which satisfy a notion we refer to as collision sensitivity*. As the name suggests, a class of predicates is collision sensitive if finding a collision in a given hash function breaks the soundness of the hash.*

Definition 22. *A class of predicates $\mathcal{P}$ is* collision sensitive *if there exists a PPT algorithm $\mathcal{A}$ such that for any pair x, x', $\Pr[\mathcal{A}(x, x') \to P : P(x) \neq P(x')] \geq 1 - \text{negl}(n)$.*

Notice that a class of predicates being reconstructing *(as per Section 3.4.2) automatically implies collision-sensitivity. Indeed, it is a stronger characteristic: Since the reconstructing learner can use a series of predicates P to determine x from all other x' with negligible probability, this already implies we can use that same series P to distinguish x from any other x' (also with negligible probability).*

We show two lower bounds for achieving a PPH for any class of collision-sensitive predicates $\mathcal{P}$:

1. *Direct-Access robust PPHs for $\mathcal{P}$ implies the existence of collision resistant hash functions (using the definition of equality PPHs).*

2. *Double-Oracle robust PPHs for $\mathcal{P}$ implies one-way functions (using techniques from [110]).*

These results follow from characterizations of PPH for the specific case of the equality *predicate in the respective models.*

On the other hand, we demonstrate an unconditional construction for the weaker notion of Evaluation-Oracle PPHs for equality, using pairwise independence. Note that existence of an unconditional construction is to be expected, as Evaluation-Oracle PPHs align with non-robust PPHs for the case of total predicates (such as equality).

3.5.1 The Equality Predicate and Collision-Sensitivity

Denote $Q_{x_2}(x_1) := [x_1 == x_2]$ the x_2-parameterized equality predicate, and denote $Q'_{x_2}(y) := \mathcal{H}.\mathsf{Eval}(h, Q_{x_2}, y)$ for a given hash function h sampled from PPH $\mathcal{H}$. One thing to notice is that finding any collision with respect to h, i.e. $h(x_1) = h(x_2)$ but $x_1 \neq x_2$, means that $Q'_{x_2}(h(x_1)) = Q'_{x_2}(h(x_2))$, and so either $Q'_{x_2}(h(x_1)) \neq Q_{x_2}(x_1)$ or $Q'_{x_2}(h(x_2)) \neq Q_{x_2}(x_2)$. This necessarily means that no matter what Q' actually computes with respect to x_2 and $h(x_1) = h(x_2)$, its output at least one of the inputs x_1, x_2 is incorrect. We leverage this together with the following reduction from PPH for any collision-sensitive predicate class to PPH for equality, in order to prove lower bounds.

Theorem 20. *If there exists a (direct-access / double-oracle / evaluation-oracle) robust PPH for any collision-sensitive predicate $\mathcal{P}$ with compression η, then there exists a (direct-access / double-oracle / evaluation-oracle, resp.) robust equality PPH with compression η.*

Proof. We will prove the contrapositive. Assume in any of our robust models that there does not exist an equality PPH with compression η. Then, for any η-compressing hash h, there exists an adversary $\mathcal{B}$ that, when playing the game corresponding to the model, can output an x_1 and x_2 such that $Q_{x_2}(x_1) \neq Q'_{x_2}(h(x_1))$.

Now, for sake of contradiction, also assume that there exists a PPH $\mathcal{H}$ for a class of collision-sensitive predicates $\mathcal{P}$. Consider the following PPH $\mathcal{H}'$ for the class of equality predicates, where $\mathcal{H}'.\mathsf{Samp} = \mathcal{H}.\mathsf{Samp}$, and where we define $\mathcal{H}'.\mathsf{Eval}(h, Q_{x_2}, h(x_1)) = 1$ if and only if $h(x_1) = h(x_2)$. Note that $\mathcal{H}'$ also has compression factor η. Because equality PPH does not exist by assumption, there exists an efficient adversary $\mathcal{B}$ who can output x_1 and x_2 such that $Q_{x_2}(x_1) \neq \mathcal{H}'.\mathsf{Eval}(h, Q_{x_2}, h(x_1))$ with non-negligible probability. By construction, it must be that $x_1 \neq x_2$. This implies $Q_{x_2}(x_1) = 0$ and therefore $\mathcal{H}'.\mathsf{Eval}(h, Q_{x_2}, h(x_1)) = 1$. From our definition of Q', this means that $h(x_1) = h(x_2)$. Now because $\mathcal{P}$ is collision-sensitive, we can use algorithm $\mathcal{A}$ on x_1, x_2 to generate a predicate $P_{cs} \in \mathcal{P}$ such that $P_{cs}(x_1) \neq P_{cs}(x_2)$. However, because $h(x_1) = h(x_2)$, $\mathcal{H}.\mathsf{Eval}(h, P_{cs}, h(x_1)) = \mathcal{H}.\mathsf{Eval}(h, P_{cs}, h(x_2))$. One of these evaluations *must* be incorrect. Therefore, any attack against the corresponding equality version of the PPH is also an attack against the collision-sensitive predicate class PPH (in any model). □

In the following subsections, we focus on characterizations of PPH for equality within our respective levels of robustness. Then in Section 3.5.5 we return to the corresponding implications for PPH for any class of collision-sensitive predicates.

3.5.2 Direct-Access Equality PPHs if and only if CRHFs

We observe that Direct-Access robust PPHs for equality are equivalent to collision-resistant hash functions (CRHFs):

Definition 23. *A family of functions $\mathcal{H} = \{h : X \to Y\}$ is a family of CRHFs if it is efficiently sampleable, efficiently evaluatable, compressing, and for any PPT adversary $\mathcal{A}$,*

$$\Pr_{h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\lambda)} [\mathcal{A}(h) \to (x_1, x_2) : h(x_1) = h(x_2) \wedge x_1 \neq x_2] \leq \mathrm{negl}(\lambda)$$

Notice that a CRHF family satisfies the definition of an equality PPH, $\mathcal{H}.\mathsf{Eval}(h, P_{x_2}, y) = (h(x_2) == y)$: an adversary who finds $\mathcal{H}.\mathsf{Eval}(h, P_{x_2}, h(x_1)) \neq (x_1 == x_2)$, violating correctness of the PPH, must have found $h(x_1) = h(x_2)$, violating the collision-resistance of the CRHF. Notice also that an equality PPH must satisfy collision-resistance: an adversary finding a collision between x_1 and x_2 can break the correctness of the PPH with either $\mathcal{H}.\mathsf{Eval}(h, P_{x_2}, h(x_1))$ or $\mathcal{H}.\mathsf{Eval}(h, P_{x_2}, h(x_2))$. Therefore, the two definitions are equivalent.

3.5.3 Double-oracle Equality PPHs if and only if OWFs

We will prove that such a hash family existing is equivalent to OWFs. This is significantly less obvious than the previous characterization of the equality PPHs using CRHFs. First we will show the obvious direction, that OWFs imply Double-oracle Equality PPHs.

Claim 13. *Suppose one-way functions exist. Then for any polynomial p, there exist Double-Oracle robust PPH families $\mathcal{H} = \{h : \{0,1\}^n \to \{0,1\}^m\}$ for equality also exist.*

Proof. OWFs imply the existence of (compressing) PRFs. Let $\mathcal{H} = \{h : \{0,1\}^n \to \{0,1\}^m\}$ be a family of PRFs. We will prove that $\mathcal{H}$ is also an equality-preserving hash robust in the Double-Oracle model.

Consider an adversary $\mathcal{A}$ that has a non-negligible advantage at finding a collision. That is, $\Pr_h\left[\mathcal{A}^{h(\cdot)} \to (x,y) : h(x) = h(y)\right] \geq \epsilon + \delta$, where δ is non-negligible. Now, let $R : \{0,1\}^n \to \{0,1\}^m$ be a truly random function. Clearly, for any $\mathcal{A}$, $\Pr_h\left[\mathcal{A}^{R(\cdot)} \to (x,y) : h(x) = h(y)\right] = \epsilon$ — no PPT algorithm can have any advantage over finding a collision in a random function beyond a guaranteed collision probability for random guessing.

Since $\mathcal{A}$ has a noticeable advantage, we can distinguish when h is a PRF and when we have a random oracle. This contradicts the definition of a PRF. Therefore, no PPT $\mathcal{A}$ should have more than a negligible advantage in producing a collision. □

Double-Oracle PPHs for Equality imply OWFs.

We will show that without OWFs, given any *compressing family of functions $\mathcal{H}$, we can find a collision given only an oracle to $h \in \mathcal{H}$ with noticeable probability. Finding a collision is equivalent to finding a pair of inputs breaking the guarantees of the PPH.*

Theorem 21. *Double-Oracle robust PPHs for equality imply OWFs.*

The proof of this theorem is in appendix 3.5.6 and is an adaptation and generalization of the proof that adversarially robust Bloom Filters require OWFs from [110]. The basic idea is to use the fact that we can invert any poly-time evaluatable function in polynomial time to reverse-engineer the randomness used in generating that specific hash function from $\mathcal{H}.\mathsf{Samp}$. Once we are able to do this, we can augment that inversion algorithm to also return a nearly random preimage. This will cause us to find a collision with noticeable probability.

3.5.4 Evaluation-Oracle PPHs for Equality with Pairwise Independence.

We note that we do not need any computational assumptions to obtain an Equality PPH in the Evaluation-Oracle model. Let $\mathcal{F}$ be a pairwise-independent hash family from m bits to n bits, with $n < m$. We will prove that $\mathcal{F}$ is secure in the Evaluation-Oracle model.

Theorem 22. *Let $\mathcal{F} = \{f : \{0,1\}^m \to \{0,1\}^n\}$ be any compressing pairwise-independent hash family. $\mathcal{F}$ is an equality-preserving hash that is robust in the Evaluation Oracle Model.*

Proof. First, note that $\mathcal{F}$ is compressing by definition. So, we will move on to proving it is secure. We will show that we can replace every query answered by the evaluation

oracle with an oracle that just returns the correct answer using a series of hybrids. Let $\mathcal{O}_{eq}$ be an oracle that returns 1 if two strings are different and 0 if they are equal.

So, let $\mathcal{A}$ be a PPT adversary and let $T = \text{poly}(n)$ be the maximum number of queries $\mathcal{A}$ can make. We will prove that $\mathcal{A}$ cannot distinguish whether he is receiving actual predicate evaluations or correct evaluations. In Hybrid 0, $\mathcal{A}$ is using $\mathcal{O}_{h.\mathsf{Eval}'}$ for all of the queries. In Hybrid t, $\mathcal{A}$ is getting answers from $\mathcal{O}_{eq}$ for the first t queries, and then gets answers from $\mathcal{O}_{h.\mathsf{Eval}'}$ for the last $T - t$ queries.

We will now show that statistically $\mathcal{A}$ cannot distinguish between Hybrid t and Hybrid $t-1$. So, $\mathcal{A}$ has made $t-1$ queries and gotten correct responses for each of them. $\mathcal{A}$'s tth query can be x_1, x_2 where $x_1 = x_2$ or $x_1 \neq x_2$. Both oracles are guaranteed to answer the same way if $x_2 = x_2$, so let's examine the case where $x_1 \neq x_2$. The probability over our choice of $f \in \mathcal{F}$ that $h(x_1) = h(x_2)$ is 2^{-n} because $\mathcal{F}$ is pairwise independent. Therefore, the probability that $\mathcal{O}_{h.\mathsf{Eval}'}$ answers differently from $\mathcal{O}_{h.\mathsf{Eval}'}$ is 2^{-n}, and $\mathcal{A}$ can only distinguish Hybrid t and $t-1$ with probability 2^{-n}.

This means that $\mathcal{A}$ can distinguish Hybrid 0 from Hybrid T with probability at most $\text{poly}(n) \cdot 2^{-n} = \text{negl}(n)$ (union bound). □

3.5.5 Collision-Sensitivity, OWFs, and CRHFs

As shown in Theorem 20, any lower bound for equality PPHs implies a lower bound for all PPHs for collision sensitive predicate classes. Therefore, we get the following two corollaries.

Corollary 6. *Let $\mathcal{P}$ be any collision-sensitive predicate class. Then any PPH for $\mathcal{P}$ in the Direct-Access model implies that CRHFs exist.*

Corollary 7. *Let $\mathcal{P}$ be any collision-sensitive predicate class. Then any PPH for $\mathcal{P}$ in the Evaluation-Oracle model implies that OWFs exist.*

3.5.6 Proof of Theorem 21: A Double-Oracle Equality PPH implies OWFs

We will show that without OWFs, given any *compressing family of functions $\mathcal{H}$, we can find a collision given only an oracle to $h \in \mathcal{H}$ using algorithm 3.5.6, and finding such a collision is equivalent to finding a pair of inputs breaking the guarantees of the PPH.*

Before our proof, we will need to define bins.

Definition 24. *A bin $B_y \subseteq \{0,1\}^n$ is defined by an element in $y \in Im(g)$ for some function g: $B_y = \{x : g(x) = y\}$.*

We will typically fix a bin and analyze the properties of that bin, so we will drop the subscript.

We will also assume that OWFs do not exist, which allows us to have non-uniform inverters.

Theorem 23 ([63])**.** *If OWFs do not exist, then weak OWFs do not exist. This means that for every function $f : \{0,1\}^n \to \{0,1\}^m$, there exists a non-uniform inverter $\mathcal{A}_f$ and a polynomial Q,*

$$\Pr_x [y \leftarrow f(x), x' \leftarrow \mathcal{A}_f(y) : f(x') = y] \geq 1 - \frac{1}{Q(n)}.$$

Here is an overview of our proof: we will use our oracle access to h and our ability to invert function to produce an approximation h' of h. We then need to have a noticeable probability of choosing an element x and getting an inverse x' from an inverter $\mathcal{A}_h$ such that neither x nor x' disagree between h and h'. The way to ensure this is to think of the output bin defined by x and bound the fraction of elements that are bad (that disagree between h and h'). Since we may have some bad elements in our bin, even if they are a small fraction, an arbitrary inverter of h' might always choose to give us inverses that disagree between h and h'. So, we will want to split the bin into sub-bins using a pairwise independent hash function $f \xleftarrow{\$} \mathcal{F}_k$ for some k – lemma 24 tells us that there exists a $k \in [n]$ so that there are very few bad elements in our sub-bin in expectation.

FindCollisions

Theorem 24. *Double-Oracle robust PPHs for equality imply OWFs.*

Proof. Throughout this proof, we will reference a few lemmas proved later. We will present this proof first to motivate the lemmas. We will prove that algorithm 3.5.6 finds collisions in any compressing function with noticeable probability.

To put this all together, let's label some events:

- InverseSuccess(IS) is the event that $\mathcal{A}_{g_k}$ outputs a correct inverse.
- Collision is the event we get a collision with algorithm 3.5.6.
- GoodApprox is the event that $\mathcal{A}_{g_k}$ produces an h' that disagrees with h on at most $1/(100n^c)$-fraction of inputs. A *bad* element is an element x such that $h(x) \neq h'(x)$.
- GoodBin is the event that the x we chose landed in a bin with at most $1/n^c$ bad elements and has more than 1 element.
- CleanSubBin is the event that after choosing f from $\mathcal{F}_k$ for $k = \max(0, i-3)$, we end up in a sub-bin with no bad elements and more than one element. Otherwise we refer to the sub-bin as dirty.

We will first assume that $\mathcal{A}_{g_k}$ succeeds and that $k = \max(0, i-3)$. We will compute the probability of failure using a union bound later, and since k is chosen independently at random of all other events, we will include that with probability $1/n < 1/(n-3)$.

We dissect the probability, over our choice of $\mathbf{x} \xleftarrow{\$} \{0,1\}^{n \times t}$, $x \xleftarrow{\$} \{0,1\}^n$, and $f \xleftarrow{\$} \mathcal{F}_k$ where $k = \max(0, i-3)$, of getting a collision *given* InverseSuccess; e.g. in

the next lines Pr[Collision] really means Pr[Collision|IS]:

$$\begin{aligned}\Pr_{\mathbf{x},x,f}[\mathsf{Collision}] &\geq \Pr[\mathsf{Collision}|\mathsf{CleanSubBin}] \cdot \Pr[\mathsf{CleanSubBin}] \\ &\geq \Pr[\mathsf{Collision}|\mathsf{CleanSubBin}] \cdot \Pr[\mathsf{CleanSubBin}|\mathsf{GoodBin}] \cdot \\ &\quad \Pr[\mathsf{GoodBin}] \\ &\geq \Pr[\mathsf{Collision}|\mathsf{CleanSubBin}] \cdot \Pr[\mathsf{CleanSubBin}|\mathsf{GoodBin}] \cdot \\ &\quad \Pr[\mathsf{GoodBin}|\mathsf{GoodApprox}] \cdot \Pr[\mathsf{GoodApprox}]\end{aligned}$$

Now, let's analyze each of these probabilities one-by-one, starting with Pr[GoodApprox|IS].

- $\Pr[\mathsf{GoodApprox}|\mathsf{IS}] \geq 99/100$.

  This is a straight application of lemma 22. We guarantee that with probability $99/100$, h' agrees with h on all but $1/p(n)$-fraction of inputs for any polynomial $p(n)$ depending on our number of queries t. Now, we have chosen $100n^c$ for our polynomial.

- $\Pr[\mathsf{GoodBin}|\mathsf{GoodApprox} \wedge \mathsf{IS}] \geq 49/100$.

  From lemma 23, we know that $99/100$-fraction of our inputs end up in a bin where at most $100/(100n^c) = 1/n^c$-fraction of the inputs disagree between h and h'. Now, we could end up in a bin with only one element, but at most $1/2$ of the elements could map to bins of size 1. This is because if we compress even just by one bit and map as many of our 2^n elements to bins with a single element in them, we can map the first $2^{n-1} - 1$ to single-element bins, but the last bin must contain the rest, and $\frac{2^{n-1}-1}{2^n} < \frac{1}{2}$. Compressing by more bits only decreases this fraction.

  Therefore, Pr[GoodBin|GoodApprox∧IS] = 1−Pr[only one element in bin or in a bad bin]. By a union bound, this gives us $1 - (1/2 + 1/100) = 49/100$.

- $\Pr[\mathsf{CleanSubBin}|\mathsf{GoodBin} \wedge \mathsf{IS}] \geq 79/100$.

  From lemma 24, we know that our sub-bin contains bad elements with probability at most $16/n^c$. Assume we have chosen c large enough so that $16/n^c \leq 1/100$. Then, from lemma 25, we know that with probability at least $8/10$, we have more than 1 element. So, Pr[CleanSubBin|GoodBin∧IS] = 1−Pr[only element in sub-bin or in a dirty su
  By a union bound this is at least $1 - (2/10 + 1/100) = 79/100$.

- $\Pr[\mathsf{Collision}|\mathsf{CleanSubBin} \wedge \mathsf{IS}] \geq \frac{1}{2}$.

  We are guaranteed to have at least two elements in our sub-bin. So, when $\mathcal{A}_{g_k}$ produces a pre-image for $h'(x)||f(x)$, then $\mathcal{A}$ has probability at most $1/2$ of giving us $x' = x$, but we are guaranteed that $h'(x') = h(x')$ since we are in a clean sub-bin.

This means,

$$\Pr[\mathsf{Collision}|\mathsf{IS}] \geq \frac{1}{2} \cdot \left(\frac{79}{100} \cdot \frac{49}{100} \cdot \frac{99}{100}\right) \geq \frac{19}{100}$$

We're almost done. We need to finish this analysis by consider the case that $\mathcal{A}_g$ does not find an inverse and that we correctly chose $k = \min(0, i-3)$. First, note that $\Pr[\neg \mathsf{IS}] \leq 1/100$ from theorem 23 because the input to $\mathcal{A}_g$ looks like a random output from g. Finally,

$$\begin{aligned}\Pr[\mathsf{Collision}] &\geq \Pr[\mathsf{Collision}|\mathsf{IS}] \cdot \Pr[\mathsf{IS}] \cdot \Pr[\ k = \min(0, i-3)\ ] \\ &\geq \frac{19}{100} \cdot \frac{99}{100} \cdot \frac{1}{n} \geq \frac{18}{100n}\end{aligned}$$

□

Helpful lemmas

Here we will go through the lemmas that make our analysis possible. We will start by showing that with polynomially many queries t, we have a 99/100 chance of getting an approximate h' that agrees on almost all queries with h.

Lemma 22. *Let $p(n)$ be any polynomial and assume OWFs do not exist. Let $t \geq (r+7)p(n)$, and we define a function $g(h, x_1, \ldots, x_t) = x_1, \ldots, x_t, h(x_1), \ldots, h(x_t)$. Let $\mathbf{x} = (x_1, \cdots, x_t) \xleftarrow{\$} \{0,1\}^{t \cdot n}$. For any non-uniform inverter $\mathcal{A}_g$ that succeeds in producing an inverse h',*

$$\Pr_{h,\mathbf{x}}\left[h' \leftarrow \mathcal{A}_g(\mathbf{x}, \mathbf{y}) : \Pr_x\left[h(x) = h'(x)\right] \geq 1 - 1/p(n)\right] \geq 98/100.$$

Proof. Let $r = n^{O(1)}$ be the number of bits required to describe a hash function in $\mathcal{H}$. Now, fix h, the hash function we have oracle-access to. We consider the following function, which we use in line 2 of algorithm 3.5.6:

$$\begin{aligned}g &: \{0,1\}^r \times \{0,1\}^{tn} \to \{0,1\}^{tn} \times \{0,1\}^{tm} \\ g &: (h, x_1, \cdots, x_t) \mapsto (x_1, \cdots, x_t, h(x_1), \cdots, h(x_t))\end{aligned}$$

Since OWFs do not exist, we have an inverter $\mathcal{A}_g$ which can invert g to get an h' on at least 99/100-fraction of possible outputs of g. First, assume that $\mathcal{A}_g$ produces a correct inverse. Now, we will bound the probability that h' differs on *more than* $1/p(n)$-fraction of inputs to h:

$$\Pr_{\mathbf{x}}\left[\forall x_i, h(x_i) = h'(x_i)\right] \leq (1 - \frac{1}{p(n)})^t.$$

Since h' has only r bits to describe itself, we can bound the probability that there even exists such an h' with a union bound:

$$\Pr_{\mathbf{x}}\left[\exists h' : \forall x_i, h(x_i) = h'(x_i)\right] \leq 2^r \left(1 - \frac{1}{p(n)}\right)^t.$$

We want this probability to be less than $\frac{1}{100}$, so we can bound the number of queries t as follows:

$$2^r \left(1 - \frac{1}{p(n)}\right)^t \leq \frac{1}{100} \iff t \geq (r + \log(100)) \cdot \frac{1}{\log\left(\frac{p(n)}{p(n)-1}\right)}$$

We notice that this ugly term $\frac{1}{\log \frac{p(n)}{p(n)-1}} \leq p(n)$ for all n. It turns out that $p(n)$ is a very good upper bound of this term: assuming $p(n)$ is increasing, for all exponents $0 \leq c < 1$, there exists n' so that for all $n > n'$, $\frac{1}{\log \frac{p(n)}{p(n)-1}} \geq p(n)^c$.

Given that $7 > \log_2(100)$, we can choose t such that

$$t \geq (r+7)p(n).$$

□

Now, we will assume that h and h' agree on all but $1/p(n)$ inputs and prove, using a simple counting argument, that $99/100$ of our inputs land in bins with at most $100/p(n)$-fraction of bad bins.

Lemma 23. *If h and h' disagree on at most $q(n) = 2^n/p(n)$, then there exists a set of bins containing $99/100$-fraction of all inputs such that each bin contains at most a $1/p'(n)$-fraction of bad inputs where $p'(n) = p(n)/100$.*

Proof. For sake of contradiction, assume that the lemma is not true: for *every* set of $99/100 \cdot 2^n$ inputs $x \in \{0,1\}^n$, at least one of the x falls into a bin B, where strictly more than $1/p'(n)$-fraction of the inputs $y \in B$ are "bad," mapping $h(y) \neq h'(y)$.

So, let us consider the set where we map as many inputs as we can to good bins, bins with $\leq 1/p'(n)$-fraction of the inputs are bad. This means that the rest of the inputs map to bins where $> 1/p'(n)$-fraction of the inputs are bad. In the best case for this, we can find good bins for $99/100 \cdot 2^n - 1$ of the inputs, but not for the last one.

Let $q' \geq 2^n/100 + 1$ be the number of inputs that are left, and therefore all map to bad bins. In fact, these q' elements fill the remaining bins exactly, so if we try counting the number of bad elements:

$$\#bad > \sum_{B \text{ is a bad bin}} \left(|B| \cdot \frac{1}{p'(n)} \right) = \frac{q'}{p'(n)}.$$

Since $q' > 2^n/100$, this implies $\#bad > 2^n/(100p'(n)) = 2^n/p(n)$. This is a contradiction since we assumed $\#bad \leq 2^n/p(n)$.

Therefore, there exists a subset $S \subset \{0,1\}^n$ where $|S| \geq 2^n/100$, and all $x \in S$ map to bins with fraction at most $100/p(n)$ of the elements in those bins are bad. □

Now, let us assume that we are in a good bin where h' agrees with h on all but $1/n^c$ elements in this bin. We will prove that with noticeable probability, we can split the bin into sub-bins, where most of them are completely clean (and in fact that we will land in a clean bin).

Lemma 24. *Let B be a bin of size between 2^i and 2^{i+1}, and at most $1/n^c$-fraction of elements $x \in B$ are bad.*

Let $k = \max(0, i-3)$ and $f : \{0,1\}^n \to \{0,1\}^k \xleftarrow{\$} \mathcal{F}$ a pairwise independent hash-function family. For an arbitrary fixed $x \in B$,

$$\Pr_{f \xleftarrow{\$} \mathcal{F}} [\nexists x' \in B \text{ so that } h(x') \neq h'(x') \wedge f(x') = f(x)] \geq \frac{16}{n^c}$$

Proof. Let $|B| = s$. By assumption, $2^i \leq s \leq 2^{i+1}$. We will be focusing on the number of bad elements in B. Let q be the number of bad elements in B. Again, by assumption, $q \leq s/n^c$.

We have two cases: $i < 4$ and $i \geq 4$. For the case that $i < 4$, we can choose c to be large enough so that $2^i/n^c < 1$. When this is the case, there are no bad elements in B, and therefore for $k = 0$, the sub-bin defined by $h'(x)||f(x) = h'(x)$ is entirely clean. In fact, for $n > 2$, we choose $c \geq 4$ and we are guaranteed this fact. When $i \geq 4$, we need to do a bit more analysis.

Fix a sub-bin B' for an arbitrary element in the image of $f \in \mathcal{F}$. Let $\hat{X} = \sum_{j=1}^{q} \hat{X}_j$ be the sum of indicator values $\hat{X}_j$ where $\hat{X}_j$ is 1 if the j^{th} *bad* element in our starting bin B is mapped to bin B'. So, $X = |B'|$ is a non-negative random element. We can bound the mean of $\hat{X}$ as $\hat{\mu} = \mathrm{E}[\hat{X}] \leq \frac{s}{n^c} \cdot \frac{1}{2^k}$. Since s is between 2^i and 2^{i+1}, $\hat{\mu} \leq \frac{16}{n^c}$. With a Markov bound, $\Pr[\hat{X} \geq 1] \leq \frac{\hat{\mu}}{1} \leq \frac{16}{n^c}$. □

Finally, we need to make sure that our sub-bin is large enough. So, we will assume that we are in a bin of size s between 2^i and 2^{i+1} and let $k = i - 3$, as in the previous theorem. We will show using the assymetric Chebyshev theorem, theorem 25, that with probability $8/10$, we have a sub-bin of at least 2 elements.

Theorem 25 (Asymetric Chebyshev)**.** *For a random variable X of unknown or assymetric distribution with mean μ and variance σ^2 and for two integers, $k_1 + k_2 = 2\mu$,*

$$\Pr\left[k_1 < X < k_2\right] \geq \frac{4\left((\mu - k_1)(\mu - k_2) - \sigma^2\right)}{(k_2 - k_1)^2}.$$

Lemma 25. *Let B be a bin of size between 2^i and 2^{i+1} for $i \geq 1$, and at most $1/n^c$-fraction of elements $x \in B$ are bad.*

Let $k = \max(0, i - 3)$ and $f : \{0,1\}^n \to \{0,1\}^k \xleftarrow{\$} \mathcal{F}$ a pairwise independent hash-function family. With probability at least $8/10$, there are at least 2 elements in a sub-bin defined by $h'(x)||f(x)$.

Proof. In this proof we consider the bin B as a whole. Let $X = \sum_{j=1}^{s} X_j$ where X_j indicates if the j^{th} element in B maps to our sub-bin B' defined by $h'(x)||f(x)$. Note that since f is pairwise independent, X is the sum of pairwise independent variables.

Again, the first case, where $i < 4$, is simple to analyze. Here $k = 0$, and we are looking at B as a whole, which by assumption $i \geq 1$ has at least 2 elements. The second case, where $i \geq 4$, requires more analysis.

Let μ be the mean of X. By linearity of expectation $\mu = \sum \mathrm{E}[X_j] = \sum \frac{1}{2^k} = \frac{s}{2^k}$. Since $k = i - 3$, $8 \leq \mu \leq 16$.

Let σ^2 be the variance of X. We compute σ^2 as follows, keeping in mind that the

X_j are pairwise independent so covariance between 2 variables is 0:

$$\begin{aligned}\sigma^2 &= \mathrm{Var}\left(\sum_{j=1}^{s} X_j\right) = \sum_{j=1}^{s} \mathrm{Var}(X_j) + \sum_{j\neq\ell} \mathrm{Cov}(X_j, X_\ell) \\ &= \sum_{j=1}^{s} \mathrm{Var}(X_j) = \sum_{j=1}^{s} \frac{1}{2^k}\left(1 - \frac{1}{2^k}\right) \\ &= \frac{s}{2^k}(1 - \frac{1}{2^k}) < \mu.\end{aligned}$$

Since we have variance and mean, we can use theorem 25, the Chebyshev inequality. If we let $k_1 = 1$ and $k_2 = 2\mu - 1$, we satisfy $k_1 + k_2 = 2\mu$, and can compute

$$\begin{aligned}\Pr[X > 1] \geq \Pr[1 < X < 2\mu - 1] &\geq \frac{4(\mu-1)(2\mu - 1 - \mu) - \sigma^2}{(2\mu - 1 - 1)} \\ = \frac{4}{4(\mu-1)^2} \cdot \left((\mu-1)^2 - \sigma^2\right) &\geq \frac{((\mu-1)^2 - \mu)}{(\mu-1)^2}.\end{aligned}$$

Recall that $8 \leq \mu \leq 16$ and this function increases with μ. So, plugging in $\mu = 8$ gives us a minimum:

$$\Pr[X > 1] \geq \frac{(8-1)^2 - 8}{(8-1)^2} = \frac{41}{49} > \frac{8}{10}.$$

□

Chapter 4

Fine-Grained Cryptography

Blurb about FGC.

4.1 Overview

4.2 Introduction

Modern cryptography has developed a variety of important cryptographic primitives, from One-Way Functions (OWFs) to Public-Key Cryptography to Obfuscation. Except for a few more limited information theoretic results [123, 46, 122], cryptography has so far required making a computational assumption, $P \neq NP$ being a baseline requirement. Barring unprecedented progress in computational complexity, such hardness hypotheses seem necessary in order to obtain most useful primitives. To alleviate this reliance on unproven assumptions, it is good to build cryptography from a variety of extremely different, believable assumptions: if a technique disproves one hypothesis, the unrelated ones might still hold. Due to this, there are many different cryptographic assumptions: on factoring, discrete logarithm, shortest vector in lattices and many more.

Unfortunately, almost all hardness assumptions used so far have the same quite stringent requirements: not only that NP is not in BPP, but that we must be able to efficiently sample polynomially-hard instances whose solution we know. Impagliazzo [77, 117] defined five worlds, which capture the state of cryptography, depending on which assumptions happen to fail. The three worlds worst for cryptography are Algorithmica (NP in BPP), Heuristica (NP is not in BPP but NP problems are easy on average) and Pessiland (there are NP problems that are hard on average but solved hard instances are hard to sample, and OWFs do not exist). This brings us to our main question.

Can we have a meaningful notion of cryptography even if we live in Pessiland (or Algorithmica or Heuristica)?

This question motivates a weaker notion of cryptography: cryptography that is secure against n^k-time bounded adversaries, for a constant k. Let us see why such

cryptography might exist even if P = NP. In complexity, for most interesting computational models, we have time hierarchy theorems that say that there are problems solvable in $O(n^2)$ time (say) that cannot be solved in $O(n^{2-\epsilon})$ time for any $\epsilon > 0$ [69, 72, 127]. In fact, such theorems exist also for the average case time complexity of problems [96]. Thus, even if P=NP, there are problems that are hard on average for specific runtimes, i.e. fine-grained *hard on average.* Can we use such hard problems to build useful cryptographic primitives?

Unfortunately, the problems from the time hierarchy theorems are difficult to work with, a common problem in the search for unconditional results. Thus, let us relax our requirements and consider hardness assumptions, but this time on the exact running time of our problems of interest. One simple approach is to consider all known constructions of Public Key Cryptography (PKC) to date and see what they imply if the hardness of the underlying problem is relaxed to be $n^{k-o(1)}$ for a fixed k (as it would be in Pessiland). Some of the known schemes are extremely efficient. For instance, the RSA and Diffie-Hellman cryptosystems immediately imply weak PKC if one changes their assumptions to be about polynomial hardness [121, 52]. However, these cryptosystems have other weaknesses – for instance, they are completely broken in a postquantum world as Shor's algorithm breaks their assumptions in essentially quadratic time [124]. Thus, it makes sense to look at the cryptosystems based on other assumptions. Unfortunately, largely because cryptography has mostly focused on the gap between polynomial and superpolynomial time, most reductions building PKC have a significant (though polynomial) overhead; many require, for example, multiple rounds of Gaussian elimination. As a simple example, the Goldreich-Levin construction for hard-core bits uses n^ω (where $\omega \in [2, 2.373)$ is the exponent of square matrix multiplication [129][60]) time and n calls to the hard-core-bit distinguisher [64]. The polynomial overhead of such reductions means that if the relevant problem is only $n^{2-o(1)}$ hard, instead of super-polynomially hard, the reduction will not work anymore and won't produce a meaningful cryptographic primitive. Moreover, reductions with fixed polynomial overheads are no longer composable in the same way when we consider weaker, polynomial gap cryptography. Thus, new, more careful cryptographic reductions are needed.

Ball et al. [15, 16] recently began to address this issue through the lens of the recently blossoming field of fine-grained complexity*. Fine-grained complexity is built upon "fine-grained" hypotheses on the (worst-case) hardness of a small number of key problems. Each of these key problems K, has a simple algorithm using a combination of textbook techniques, running in time $T(n)$ on instances of size n, in, say, the RAM model of computation. However, despite decades of research, no $\tilde{O}(T(n)^{1-\epsilon})$ algorithm is known for any $\epsilon > 0$ (note that the tilde suppresses sub-polynomial factors). The fine-grained hypothesis for K is then that K requires $T(n)^{1-o(1)}$ time in the RAM model of computation. Some of the main hypotheses in fine-grained complexity (see [128]) set K to be CNF-SAT (with $T(n) = 2^n$, where n is the number of variables), or the k-Sum problem (with $T(n) = n^{\lceil k/2 \rceil}$), or the All-Pairs Shortest Paths problem (with $T(n) = n^3$ where n is the number of vertices), or one of several versions of the k-Clique problem in weighted graphs. Fine-grained uses fine-grained reductions between problems in a very tight way (see [128]): if problem A has requires running*

time $a(n)^{1-o(1)}$, and one obtains an $(a(n), b(n))$-fine-grained reduction from A to B, then problem B needs runtime $b(n)^{1-o(1)}$. Using such reductions, one can obtain strong lower bounds for many problems, conditioned on one of the few key hypotheses.

The main question that Ball et al. set out to answer is: Can one use fine-grained reductions from the hard problems from fine-grained complexity to build useful cryptographic primitives? *Their work produced worst-case to average-case fine-grained reductions from key problems to new algebraic average case problems. From these new problems, Ball et al. were able to construct fine-grained proofs of work, but they were not able to obtain stronger cryptographic primitives such as fine-grained one-way-functions or public key encryption. In fact, they gave a barrier for their approach: extending their approach would falsify the Nondeterministic Strong Exponential Time Hypothesis (NSETH) of Carmosino et al. [35]. Because of this barrier, one would either need to develop brand new techniques, or use a different hardness assumption.*

What kind of hardness assumptions can be used to obtain public-key cryptography (PKC) even in Pessiland?

A great type of theorem to address this would be: for every problem P that requires $n^{k-o(1)}$ time on average, one can construct a public-key exchange (say), for which Alice and Bob can exchange a $\lg(n)$ bit key in time $O(n^{ak})$, whereas Eve must take $n^{(a+g)k-o(1)}$ time to learn Alice and Bob's key, where g is large, and a is small. As a byproduct of such a theorem, one can obtain not just OWFs, but even PKC in Pessiland under fine-grained assumptions via the results of Ball et al. Of course, due to the limitations given by Ball et al. such an ideal theorem would have to refute NSETH, and hence would be at the very least difficult to prove. Thus, let us relax our goal, and ask

What properties are sufficient for a fine-grained average-case assumption so that it implies fine-grained PKC?

If we could at least resolve this question, then we could focus our search for worst-case to average-case reductions in a useful way.

4.2.1 Our contributions

Our main result is a fine-grained key-exchange that can be formed from any problem that meets three structural conditions in the word-RAM model of computation. This addresses the question of what properties are sufficient to produce fine-grained Public Key Encryption schemes (PKEs).

For our key exchange, we describe a set of properties, and any problem that has those properties implies a polynomial gap PKE. An informal statement of our main theorem is as follows.

Theorem. *[Fine-Grained Key-Exchange (informal)] Let P be a computational problem for which a random instance can be generated in $O(n^g)$ time for some g, and that requires $n^{k-o(1)}$ time to be solved on average for some fixed $k > g$. Additionally,*

let P have three key structural properties of interest: (1) "plantable": we can generate a random-looking instance, choosing either to have or not to have a solution in the instance, and if there is a solution, we know what/where it is; (2) "average-case list-hard": given a list of n random instances of the problem, returning which one of the instances has a solution requires essentially solving all instances; (3) "splittable": when given an instance with a solution, we can split it in $O(n^g)$ time into two slightly smaller instances that both have solutions.

Then a public key-exchange can be built such that Alice and Bob exchange a $\lg(n)$ bit key in time n^{2k-g}, where as Eve must take $\tilde{\Omega}(n^{3k-2g})$ time to learn Alice and Bob's key.

Notice that as long as there is a gap between the time to generate a random instance and the time to solve an instance on average, there is a gap between $N = n^{2k-g}$ and $n^{3k-2g} = N^{3/2-1/(4(k/g)-2)}$ and the latter goes to $N^{3/2}$, as k/g grows. The key exchange requires no interaction, and we get a fine-grained *public key cryptosystem. While our key exchange construction provides a relatively small gap between the adversary and the honest parties ($O(N^{1.5})$ vs $O(N)$), the techniques required to prove security of this scheme are novel and the result is generic as long as the three assumptions are satisfied. In fact, we will show an alternate method to achieve a gap approaching $O(N^2)$ in the full version of this paper.*

Our main result above is stated formally and in more generality in Theorem 32. We will explain the formal meaning of our structural properties plantable, average-case list-hard, *and* splittable *later.*

We also investigate what plausible average-case assumptions one might be able to make about the key problems from fine-grained complexity so that the three properties from our theorem would be satisfied. We consider the Zero-k-Clique problem as it is one of the hardest worst-case problems in fine-grained complexity. For instance, it is known that if Zero-3-Clique is in $O(n^{3-\epsilon})$ time for some $\epsilon > 0$, then both the 3-Sum and the APSP hypotheses are violated [128, 131]. It is important to note that while fine-grained problems like Zero-k-Clique and k-Sum are suspected to take a certain amount of time in the worst case, when making these assumptions for any constant k does not seem to imply $P \neq NP$ since all of these problems are still solveable in polynomial time.[1]

An instance of Zero-k-Clique is a complete k-partite graph G, where each edge is given a weight in the range $[0, R-1]$ for some integer R. The problem asks whether there is a k-clique in G whose edge weights sum to 0, modulo R. A standard fine-grained assumption (see e.g. [128]) is that in the worst case, for large enough R, say $R \geq 10n^{4k}$, Zero-k-Clique requires $n^{k-o(1)}$ time to solve. Zero-k-Clique has no non-trivial average-case algorithms for natural distributions (uniform for a range of parameters, similar to k-Sum and Subset Sum). Thus, Zero-k-Clique is a natural candidate for an average-case fine-grained hard problem.

[1] Assuming the hardness of these problems for more general k will imply $P \neq NP$, but that is not the focus of our work.

Our other contribution addresses an open question from Ball et al.: can a fine-grained one-way function be constructed from worst case assumptions? While we do not fully achieve this, we generate new plausible average-case assumptions from fine-grained problems that imply fine-grained one-way functions.

4.2.2 Previous Works

There has been much prior work leading up to our results. First, there are a few results using assumptions from fine-grained complexity and applying them to cryptography. Second, there has been work with the kind of assumptions that we will be using.

Fine-Grained Cryptography

Ball et al. [15, 16] produce fine-grained wost-case to average-case reductions. Ball et al. leave an open problem of producing a one-way-function from a worst case assumption. They prove that from some fine-grained assumptions building a one-way-function would falsify NSETH [35][15]. We avoid their barrier in this paper by producing a construction of both fine-grained OWFs and fine-grained PKE from an average-case *assumption.*

Fine-Grained Key Exchanges. *Fine-grained cryptography is a relatively unexplored area, even though it had its start in the 1970's with Merkle puzzles: the gap between honestly participating in the protocol versus breaking the security guarantee was only quadratic [104]. Merkle originally did not describe a plausible hardness assumption under which the security of the key exchange can be based. 30 years later, Biham, Goren, and Ishai showed how to implement Merkle puzzles by making an assumption of the existence of either a random oracle or an exponential gap one way function [27]. That is, Merkle puzzles were built under the assumption that a one-way function exists which takes time $2^{n(1/2+\delta)}$ to invert for some $\delta > 0$. So while prior work indeed succeeded in building a fine-grained key-exchange, it needed a very strong variant of OWFs to exist. It is thus very interesting to obtain fine-grained public key encryption schemes based on a fine-grained assumption (that might even work in Pessiland and below).*

Another notion of Fine-Grained Cryptography. *In 2016, work by Degwekar, Vaikuntanathan, and Vasudevan [51] discussed fine-grained complexity with respect to both honest parties and adversaries restricted to certain circuit classes. They obtained constructions for some cryptographic primitives (including PKE) when restricting an adversary to a certain circuit class. From the assumption* $\mathsf{NC1} \neq \oplus L/\mathrm{poly}$ *they show Alice and Bob can be in* $AC^0[2]$ *while being secure against* $\mathsf{NC1}$ *adversaries. While [51] obtains some unconditional constructions, their security relies on the circuit complexity of the adversary, and does not apply to arbitrary time-bounded adversaries as is usually the case in cryptography. That is, this restricts the types of algorithms an adversary is allowed to use beyond just how much runtime these algorithms can have. It would be interesting to get similar results in the low-polynomial time regime,*

Paper	Assumptions	Crypto	Runtime	Power of Adversary
[104]	Random Oracles*	Key Exchange	$O(N)$	$O(N^2)$
[27]	Exponentially-Strong OWFs	Key Exchange	$O(N)$	$O(N^2)$
[16]	WC 3-Sum, OV, APSP, or SETH	Proof of Work	$O(N^2)$	N/A
[This work]	Zero-k-Clique or k-Sum	OWFs, Key Exch. & PKE	$O(N)$ $O(N)$	$O(N^{1+\delta})$ $O(N^{1.5-\delta})$
[51]	$\mathsf{NC1} \neq \oplus L/\text{poly}$	OWFs, and PRGs with sublinear stretch, CRHFs, and PKE	$\mathsf{NC1}$	$\mathsf{NC1}$
	$\mathsf{NC1} \neq \oplus L/\text{poly}$	PKE and CRHFs	$\mathsf{AC}^0[2]$	$\mathsf{NC1}$
	Unconditional	PRGs with poly stretch, Symmetric encryption, and CRHFs	AC^0	AC^0

Figure 4-1: A table of previous works' results in this area. There have been several results characterizing different aspects of fine-grained cryptography. *It was [27] who showed that Merkle's construction could be realized with a random oracle. However, Merkle presented the construction.

without restricting an adversary to a certain circuit class. Our results achieve this, though not unconditionally.

Tight Security Reductions and Fine-Grained Crypto. *Another area the world of fine-grained cryptography collides with is that of tight security reductions in cryptography. Bellare et.al. coined the term "concrete" security reductions in [24, 22]. Concrete security reductions are parametrized by time (t), queries (q), size (s), and success probability (ϵ). This line of work tracks how a reduction from a problem to a construction of some cryptographic primitive effects the four parameters of interest. This started a rich field of study connecting theory to practical cryptographic primitives (such as PRFs, different instantiations of symmetric encryption, and even IBE for example [20, 21, 87, 25]). In fine-grained reductions we also need to track exactly how our adversary's advantage changes throughout our reductions, however, we also track the running time of the honest parties. So, unlike in the concrete security literature, when the hard problems are polynomially hard (perhaps because $P = NP$), we can track the gap in running times between the honest and dishonest parties. This allows us to build one way functions and public key cryptosystems when the hard problems we are given are only polynomially hard.*

Similar Assumptions

This paper uses hypotheses on the running times of problems that, while solvable in polynomial time, are variants of natural NP-hard problems, in which the size of the solution is a fixed constant. For instance, k-Sum is the variant of Subset Sum, where we are given n numbers and we need to find exactly k elements that sum to a given target, and Zero-k-Clique is the variant of Zero-Clique, in which we are given a graph and we need to find exactly k nodes that form a clique whose edge weights sum to zero.

With respect to Subset Sum, Impagliazzo and Naor showed how to directly obtain OWFs and PRGs assuming that Subset Sum is hard on average [78]. The OWF is $f(\mathbf{a}, \mathbf{s}) = (\mathbf{a}, \mathbf{a} \cdot \mathbf{s})$*, where* $\mathbf{a}$ *is the list of elements (chosen uniformly at random from the range R) and* $s \in \{0,1\}^n$ *represents the set of elements we add together. In addition to Subset Sum, OWFs have also been constructed from planted Clique, SAT, and Learning-Parity with Noise [98, 83]. The constructions from the book of Lindell and the chapter written by Barak [98] come from a definition of a "plantable" NP-hard problem that is assumed to be hard on average.*

Although our OWFs are equivalent to scaled-down, polynomial-time solvable characterizations of these problems, we also formalize the property that allows us to get these fine-grained OWFs (plantability). We combine these NP constructions and formalizations to lay the groundwork for fine-grained cryptography.

In the public-key setting, there has been relatively recent work taking NP-hard problems and directly constructing public-key cryptosystems [9]. They take a problem that is NP-hard in its worst case and come up with an average-case assumption that works well for their constructions. Our approach is similar, and we also provide evidence for why our assumptions are correct.

In recent work, Subset Sum was also shown to directly imply public-key cryptography [101]. The construction takes ideas from Regev's LWE construction [118], turning a vector of subset sum elements into a matrix by writing each element out base q in a column. The subset is still represented by a 0-1 matrix, and error is handled by the lack of carrying digits. It is not clear how to directly translate this construction into the fine-grained world. First, directly converting from Subset Sum to k-Sum just significantly weakens the security without added benefit. More importantly, the security reduction has significant polynomial overhead, and would not apply in a very pessimistic Pessiland where random planted Subset Sum instances can be solved in quadratic time, say.

While it would be interesting to reanalyze the time-complexity of this construction (and others) in a fine-grained way, this is not the focus of our work. Our goal is to obtain novel cryptographic approaches exploiting the fine-grained nature of the problems, going beyond just recasting normal cryptography in the fine-grained world, and obtaining somewhat generic constructions.

4.2.3 Technical Overview

Here we will go into a bit more technical detail in describing our results. First, we need to describe our hardness assumptions. Then, we will show how to use them for our fine-grained key exchange, and finally, we will talk briefly about fine-grained OWFs and hardcore bits.

Our Hardness Assumption

We generate a series of properties where if a problem has these properties then a fine-grained public key-exchange can be built.

One property we require is that the problem is hard on average, in a fine-grained sense. Intuitively, a problem is average case indistinguishably hard if given an instance that is drawn with probability $1/2$ *from instances with no solutions and with probability* $1/2$ *from instances with one solution, it is computationally hard on average to distinguish whether the instance has* 0 *or* 1 *solutions. The rest of the properties are structural; we need a problem that is* plantable*,* average-case list-hard*, and* splittable*. Informally,*

- *The plantable property roughly says that one can efficiently choose to generate either an instance without a solution or one with a solution, knowing where the solution is;*
- *The average case list-hard property says that if one is given a list of instances where all but one of them are drawn uniformly over instances with no solutions, and a random one of them is actually drawn uniformly from instances with one solution, then it is computationally hard to find the instance with a solution;*
- *Finally, the splittable property says that one can generate from one average case instance, two new average case instances that have the same number of solutions as the original one.*

These are natural properties for problems and hypotheses to have. We will demonstrate in Section 4.9.3 that Zero-k-Clique has all of these properties. We need our problem to have all three of these qualities for the key exchange. For our one-way function constructions we only need the problem to be plantable.

The structural properties are quite generic, and in principle, there could be many problems that satisfy them. We exhibit one: the Zero-k-Clique problem.

Because no known algorithmic techniques seem to solve Zero-k-Clique even when the weights are selected independently uniformly at random from $[0, cn^k]$ *for a constant* c*, folklore intuition dictates that the problem might be hard on average for this distribution: here, the expected number of* k*-Cliques is* $\Theta(1)$*, and solving the decision problem correctly on a large enough fraction of the random instances seems difficult. This intuition was formally proposed by Pettie [114] for the very related* k*-Sum problem which we also consider.*

We show that the Zero-k-Clique problem, together with the assumption that it is fine-grained hard to solve on average, satisfies all of our structural properties, and

thus, using our main theorem, one can obtain a fine-grained key exchange based on Zero-k-Clique.

Key Exchange Assumption. We assume that when given a complete k-partite graph with kn nodes and random weights $[0, R-1]$, $R = \Omega(n^k)$, *any adversary running in time* $n^{k-\Omega(1)}$ *time cannot distinguish an instance with a zero-k-clique solution from one without with more than* $2/3$ *chance of success. In more detail, consider a distribution where with probability* $1/2$ *one generates a random instance of size n with no solutions, and with probability* $1/2$ *one generates a random instance of size n with exactly one solution. (We later tie in this distribution to our original uniform distribution.) Then, consider an algorithm that can determine with probability* $2/3$ *(over the distribution of instances) whether the problem has a solution or not. We make the conjecture that such a* $2/3$*-probability distinguishing algorithm for Zero-k-Clique, which can also exhibit the unique zero clique whenever a solution exists, requires time* $n^{k-o(1)}$.

Public Key Exchange

So, what does the existence of a problem with our three properties, plantable*,* average-case list-hard*, and* splittable*, imply?*

The intuitive statement of our main theorem is that, if a problem has the three properties, and is n^k *hard to solve on average and can be generated in* n^g *time (for Zero-k-Clique* $g = 2$*), then a key exchange exists that takes* $O(N)$ *time for Alice and Bob to execute, and requires an eavesdropper Eve* $\tilde{\Omega}(N^{(3k-2g)/(2k-g)})$ *time to break. When* $k > g$ *Eve takes super linear time in terms of* N*. When* $k = 3$ *and* $g = 2$*, an important case for the Zero-k-Clique problem, Eve requires* $\tilde{\Omega}(N^{5/4})$ *time.*

For the rest of this overview we will describe our construction with the problem Zero-k-Clique.

To describe how we get our key exchange, it is first helpful to consider Merkle Puzzles [104, 27, 17]. The idea is simple: let f *be a one way permutation over n bits (so a range of* 2^n *values) requires* $2^{n(\frac{1}{2}+\epsilon)}$ *time to invert for some constant* $\epsilon > 0$*. Then, Alice and Bob could exchange a key by each computing* $f(v)$ *on* $10 \cdot 2^{n/2}$ *random element* $v \in [2^n]$ *and sending those values* $f(v)$ *to each other. With .9 probability, Alice and Bob would agree on at least one pre-image,* v*. It would take an eavesdropper Eve* $\Omega(2^{n(\frac{1}{2}+\epsilon)})$ *time before she would be able to find the* v *agreed upon by Alice and Bob. So, while Alice and Bob must take* $O(2^{n/2})$ *time, Eve must take* $O(2^{n(\frac{1}{2}+\epsilon)})$ *time to break it.*

Our construction will take on a similar form: Alice and Bob will send several problems to each other, and some of them will have planted solutions. By matching up where they both put solutions, they get a key exchange.

Concretely, Alice and Bob will exchange m *instances of the Zero-k-Clique problem and in* $\sqrt{m}$ *of them (chosen at random), plant solutions. The other* $m - \sqrt{m}$ *will not have solutions (except with some small probability). These* m *problems will be indexed, and we expect Alice and Bob to have both planted a solution in the same index. Alice can check her* $\sqrt{m}$ *indices against Bob's, while Bob checks his, and by the end, with constant probability, they will agree on a single index as a key. In the end, Alice and Bob require* $O(mn^g + \sqrt{m}n^k)$ *time to exchange this index. Eve must take time* $\tilde{\Omega}(n^k m)$*. When* $m = n^{2k-2g}$*, Alice and Bob take* $O(n^{2k-g})$ *time and Eve*

takes $\tilde{\Omega}(n^{3k-2g})$. We therefore get some gap between the running time of Alice and Bob as compared to Eve for any value of $k \geq g$. Furthermore, for all $\delta > 0$ there exists some large enough k such that the difference in running time is at least $O(T(n))$ time for Alice and Bob and $\tilde{\Omega}(T(n)^{1.5-\delta})$ time for Eve. Theorem 32 is the formal theorem statement.

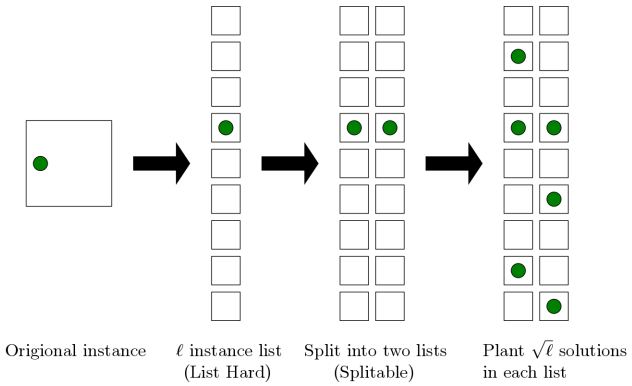

Figure 4-2: A depiction of our reduction showing hardness for our fine-grained key exchange.

To show hardness for this construction we combine techniques from both fine-grained complexity and cryptography (see Figure 4-2). We take a single instance and use a self-reduction to produce a list of ℓ instances where one has a solution whp if the original instance has a solution. In our reductions ℓ will be polynomial in the input size. Then, we take this list and produce two lists that have a solution in the same location with high probability if the original instance has a solution. Finally, we plant $\sqrt{\ell}$ solutions into the list, to simulate Alice and Bob's random solution planting.

One Way Functions *First, and informally, a fine-grained OWF is a function on n bits that requires $\tilde{O}(T(n)^{1-\delta})$ time to evaluate for some constant $\delta > 0$, and if any adversary attempts to invert f in time $\tilde{O}(T(n)^{1-\delta'})$ for* any *constant $\delta' > 0$, she only succeeds with probability at most $\epsilon(n)$, where ϵ is considered "insignificant."*

Ball et al. [15] defined fine-grained OWFs, keeping track of the time required to invert and the probability of inversion in two separate parameters. We streamline this definition by fixing the probability an adversary inverts too an insignificant function of input size, which we define in Section 4.3.

For this overview, we will focus on the intuition of using specific problems k-Sum-R (k-Sum modulo R) or Zero-k-Clique-R (Zero-k-Clique modulo R) to get fine-grained OWFs, though in section 4.7, we construct fine-grained OWFs from a general class of problems. Let N be the size of the input to these problems. Note that if R is too small (e.g. constant), then these problems are solvable quickly and the assumptions we are using are false. So, we will assume $R = \Omega(n^k)$.

OWF Assumptions. Much like for our key exchange, our assumptions are about the difficulty of distinguishing an instance of k-Sum or Zero-k-Clique with probability more than $2/3$ in time faster than $n^{k/2}$ or n^k respectively. Formally, randomly generating a k-Sum-R instance is creating a k lists of size n with values randomly chosen

from $[0, R-1]$. *Recall that a random Zero-k-Clique instance is a complete k-partite graph where weights are randomly chosen from* $[0, R-1]$. *Our 'weak' k-Sum-R and Zero-k-Clique-R assumptions state that for any algorithm running in* $O(n)$ *time, it cannot distinguish between a randomly generated instance with a planted solution and one without with probability greater than* $2/3$.

Note that these assumptions are much weaker than the previously described key-exchange assumption, where we allowed the adversary $O(n^{k-\Omega(1)})$ *time instead of just super-linear.*

Theorem 26 (Fine-Grained OWFs (informal))**.** *If for some constant* $\delta > 0$ *and range* $R = \Omega(n^k)$ *either k-Sum-R requires* $\Omega(N^{1+\delta})$ *time to solve with probability* $> 2/3$ *or Zero-k-Clique-R requires* $\Omega(N^{(1+\delta)})$ *time to solve with probability* $> 2/3$ *then a fine-grained OWF exists.*

The formal theorem is Theorem 29 and is proved in Appendix 4.7.2.

Intuitively our construction of a fine-grained OWF runs a planting procedure on a random instance in time $O(N)$. *By our assumptions finding this solution takes time* $\Omega(N^{1+\delta})$ *for some constant* $\delta > 0$, *and thus inverting this OWF takes* $\Omega(N^{1+\delta})$.

We also get a notion of hardcore bits from this. Unlike in traditional crypto, we can't immediately use Goldreich-Levin's hardcore bit construction [64]. Given a function on N bits, the construction requires at least $\Omega(N)$ *calls to the adversary who claims to invert the hardcore bit. When one is seeking super-polynomial gaps between computation and inversion of a function, factors of N can be ignored. However, in the fine-grained setting, factors of N can completely eliminate the gap between computation and inversion, and so having a notion of fine-grained hardcore bits is interesting.*

We show that for our concrete constructions of fine-grained OWFs, there is a subset of the input of size $O(\lg(N))$ *(or any sub-polynomial function) which itself requires* $\Omega(N^{1+\delta})$ *time to invert. From this subset of bits we can use Goldreich-Levin's hardcore bit construction, only losing a factor of* $N^{o(1)}$ *which is acceptable in the fine-grained setting.*

Theorem 27 (Hardcore Bits (informal))**.** *If for some constant* $\delta > 0$ *and range* $R = \Omega(n^k)$ *either k-Sum-R requires* $\Omega(N^{1+\delta})$ *time to solve with probability* $> 2/3$ *or Zero-k-Clique-R requires* $\Omega(N^{1+\delta})$ *time to solve with probability* $> 2/3$ *then a fine-grained OWF exists with a hardcore bit that can not be guessed with probability greater than* $\frac{1}{2} + 1/q(n)$ *for any* $q(n) = n^{o(1)}$.

The formal theorem is Theorem 30 and is proved in Appendix 4.7.3.

Intuitively, solutions for k-Sum-R and Zero-k-Clique-R can be described in $O(\log(n))$ *bits — we just list the locations of the solution. Given a solution for the problem, we can just change one of the weights and use the solution location to produce a correct preimage. So, now using Goldreich-Levin, we only need to make* $O(\log(n))$ *queries during the security reduction.*

4.2.4 Organization of Paper

In section 4.3 we define our notions of fine-grained crypto primitives, including fine-grained OWFs, fine-grained hardcore bits, and fine-grained key exchanges. In section 4.5, we describe a few classes of general assumptions (plantable, splittable, and average-case list hard), and then describe the concrete fine-grained assumptions we use (k-Sum and Zero-k-Clique). Next, in section 4.6 we show that the concrete assumptions we made imply certain subsets of the general assumptions. In section 4.8, we show that using an assumption that is plantable, splittable, and average-case list hard, we can construct a fine-grained key exchange.

In the supplementary appendix section 4.7, we show how to use a plantable problem to get a fine-grained OWF. In supplementary materials section 4.9 we show that Zero-k-Clique has all of the desired properties (plantable, splittable, and average-case list hard).

4.3 Preliminaries: Model of Computation and Definitions

The running times of all algorithms are analyzed in the word-RAM model of computation, where simple operations such as $+, -, \cdot$, bit-shifting, and memory access all require a single time-step.

Just as in normal exponential-gap cryptography we have a notion of probabilistic polynomial-time (PPT) adversaries, we can similarly define an adversary that runs in time less than expected for our fine-grained polynomial-time solvable problems. This notion is something we call probabilistic fine-grained time (or PFT*). Using this notion makes it easier to define things like OWFs and doesn't require carrying around time parameters through every reduction.*

Definition 25. *An algorithm $\mathcal{A}$ is an $T(n)$ probabilistic fine-grained time, $\mathsf{PFT}_{T(n)}$, algorithm if there exists a constant $\delta > 0$ such that $\mathcal{A}$ runs in time $O(T(n)^{1-\delta})$.*

Note that in this definition, assuming $T(n) = \Omega(n)$, any sub-polynomial factors can be absorbed into δ.

Additionally, we will want a notion of negligibility *that cryptography has. Recall that a function* $\mathrm{negl}(n)$ *is negligible if for all polynomials $Q(n)$ and sufficiently large n,* $\mathrm{negl}(n) < 1/Q(n)$*. We will have a similar notion here, but we will use the words* significant *and* insignificant *corresponding to non-negligible and negligible respectively.*

Definition 26. *A function* $\mathsf{sig}(n)$ *is* significant *if*

$$\mathsf{sig}(n) \geq \frac{1}{p(n)}$$

for all polynomials p. A function $\mathsf{insig}(n)$ *is* insignificant *if for all significant functions* $\mathsf{sig}(n)$ *and sufficiently large n,*

$$\mathsf{insig}(n) < \mathsf{sig}(n).$$

Note that for every polynomial f, $1/f(n)$ is insignificant. Also notice that if a probability is significant for an event to occur after some process, then we only need to run that process a sub-polynomial number of times before the event will happen almost certainly. This means our run-time doesn't increase even in a fine-grained sense; i.e. we can boost the probability of success of a randomized algorithm running in $\tilde{O}(T(n))$ from $1/\log(n)$ to $O(1)$ just by repeating it $O(\log(n))$ times, and still run in $\tilde{O}(T(n))$ time (note that ' ~ ' suppresses all sub-polynomial factors in this work).

4.3.1 Fine-Grained Symmetric Crypto Primitives

Ball et all defined fine-grained one-way functions (OWFs) in their work from 2017 [15]. They parameterize their OWFs with two functions: an inversion-time function $T(n)$ (how long it takes to invert *the function on n bits), and an probability-of-inversion function ϵ; given $T(n)^{1-\delta'}$ time, the probability any adversary can invert is $\epsilon(T(n)^{1-\delta'})$. The computation time is implicitly defined to be anything noticeably less than the time to invert: there exists a $\delta > 0$ and algorithm running in time $T(n)^{1-\delta}$ such that the algorithm can evaluate f.*

Definition 27 ((δ,ϵ)-one-way functions)**.** *A function $f : \{0,1\}^* \to \{0,1\}^*$ is* (δ,ϵ)-one-way *if, for some $\delta > 0$, it can be evaluated on n bits in $O(T(n)^{1-\delta})$ time, but for any $\delta' > 0$ and for any adversary $\mathcal{A}$ running in $O(T(n)^{1-\delta'})$ time and all sufficiently large n,*

$$\Pr_{x \leftarrow \{0,1\}^n}\left[\mathcal{A}(f(x)) \in f^{-1}(f(x))\right] \leq \epsilon(n,\delta).$$

Using our notation of $\mathsf{PFT}_{T(n)}$, we will similarly define OWFs, but with one fewer parameter. We will only be caring about $T(n)$, the time to invert, and assume that the probability an adversary running in time less than $T(n)$ inverts with less time is insignificant. We will show later, in section 4.7, that we can compile fine-grained one-way functions with probability of inversion $\epsilon \leq 1 - \frac{1}{n^{o(1)}}$ into ones with insignificant probability of inversion. So, it makes sense to drop this parameter in most cases.

Definition 28. *A function $f : \{0,1\}^* \to \{0,1\}^*$ is $T(n)$ fine-grained one-way (is an $T(n)$-FGOWF) if there exists a constant $\delta > 0$ such that it takes time $T(n)^{1-\delta}$ to evaluate f on any input, and there exists a function $\epsilon(n) \in \mathsf{insig}(n)$, and for all $\mathsf{PFT}_{T(n)}$ adversaries $\mathcal{A}$,*

$$\Pr_{x \leftarrow \{0,1\}^n}\left[\mathcal{A}(f(x)) \in f^{-1}(f(x))\right] \leq \epsilon(n).$$

With traditional notions of cryptography there was always an exponential or at least super-polynomial gap between the amount of time required to evaluate and invert one-way functions. In the fine-grained setting we have a polynomial gap *to consider.*

Definition 29. *The (relative)* gap *of an $T(n)$ fine-grained one-way function f is the constant $\delta > 0$ such that it takes $T(n)^{1-\delta}$ to compute f but for all $\mathsf{PFT}_{T(n)}$ adversaries $\mathcal{A}$,*

$$\Pr_{x \leftarrow \{0,1\}^n}\left[\mathcal{A}(f(x)) \in f^{-1}(f(x))\right] \leq \mathsf{insig}(n).$$

4.3.2 Fine-Grained Asymmetric Crypto Primitives

In this paper, we will propose a fine-grained key exchange. First, we will show how to do it in an interactive manner, and then remove the interaction. Removing this interaction means that it implies fine-grained public key encryption! Here we will define both of these notions: a fine-grained non-interactive key exchange, and a fine-grained, CPA-secure public-key cryptosystem.

First, consider the definition of a key exchange, with interaction. This definition is modified from [27] to match our notation. We will be referring to a transcript generated by Alice and Bob and the randomness they used to generate it as a "random transcript".

Definition 30 (Fine-Grained Key Exchange)**.** *A* $(T(n), \alpha, \gamma)$-*FG-KeyExchange is a protocol,* Π*, between two parties* A *and* B *such that the following properties hold*

- *Correctness. At the end of the protocol,* A *and* B *output the same bit* ($b_A = b_B$) *except with probability* γ*;*

$$\Pr_{\Pi,A,B}[b_A = b_B] \geq 1 - \gamma$$

  This probability is taken over the randomness of the protocol, A*, and* B*.*

- *Efficiency. There exists a constant* $\delta > 0$ *such that the protocol for both parties takes time* $\tilde{O}(T(n)^{1-\delta})$*.*

- *Security. Over the randomness of* Π*,* A*, and* B*, we have that for all* $\mathsf{PFT}_{T(n)}$ *eavesdroppers* E *has advantage* α *of guessing the shared key after seeing a random transcript. Where a transcript of the protocol* Π *is denoted* $\Pi(A, B)$*.*

$$\Pr_{A,B}[E(\Pi(A, B)) = b_B] \leq \frac{1}{2} + \alpha$$

A Strong $(T(n))$-*FG-KeyExchange is a* $(T(n), \alpha, \gamma)$-*FG-KeyExchange where* α *and* γ *are insignificant. The key exchange is considered* weak *if it is not strong.*

This particular security guarantee protects against chosen plaintext attacks. But first, we need to define what we mean by a fine-grained public key cryptosystem.

Definition 31. *An* $T(n)$-fine-grained public-key cryptosystem *has the following three algorithms.*

KeyGen(1^κ) *Outputs a public-secret key pair* (pk, sk)*.*

Encrypt(pk, m) *Outputs an encryption of* m*,* c*.*

Decrypt(sk, c) *Outputs a decryption of* c*,* m*.*

These algorithms must have the following properties:

- *They are efficient. There exists a constant $\delta > 0$ such that all three algorithms run in time $O\left(T(n)^{1-\delta}\right)$.*
- *They are correct. For all messages m,*

$$\Pr_{\mathsf{KeyGen},\mathsf{Encrypt},\mathsf{Decrypt}}[\mathsf{Decrypt}(sk, \mathsf{Encrypt}(pk, m)) = m | (pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)] \geq 1 - \mathsf{insig}(n).$$

The cryptosystem is CPA-secure *if any $\mathsf{PFT}_{T(n)}$ adversary $\mathcal{A}$ has an insignificant advantage in winning the following game:*

1. *Setup. A challenger $\mathcal{C}$ runs $\mathsf{KeyGen}(1^n)$ to get a pair of keys, (pk, sk), and sends pk to $\mathcal{A}$.*
2. *Challenge. $\mathcal{A}$ gives two messages m_0 and m_1 to the challenger. The challenger chooses a random bit $b \xleftarrow{\$} \{0,1\}$ and returns $c \leftarrow \mathsf{Encrypt}(pk, m_b)$ to $\mathcal{A}$.*
3. *Guess. $\mathcal{A}$ outputs a guess b' and wins if $b' = b$.*

4.4 Techniques

4.5 Average Case Assumptions

Below we will describe four general properties so that any assumed-to-be-hard problem that satisfies them can be used in our later constructions of one-way functions and cryptographic key exchanges. We will also propose two concrete problems with believable fine-grained hardness assumptions on it, and we will prove that these problems satisfy some, if not all, of our general properties.

Let us consider a search or decision problem P. Any instance of P could potentially have multiple witnesses/solutions. We will restrict our attention only to those instances with no solutions or with exactly one solution. We define the natural uniform distributions over these instances below.

Definition 32 (General Distributions)**.** *Fix a size n and a search problem P. Define $D_0(P, n)$ as the uniform distribution over the set S_0, the set of all P-instances of size n that have no solutions/witnesses. Similarly, let $D_1(P, n)$ denote the uniform distribution over the set S_1, the set of all P-instances of size n that have exactly one unique solution/witness. When P and n are clear from the context, we simply use D_0 and D_1.*

4.5.1 General Useful Properties

We now turn our attention to defining the four properties that a fine-grained hard problem needs to have, in order for our constructions to work with it.

To be maximally general, we present definitions often with more than one parameter. The four properties are: average case indistinguishably hard, plantable, average case list-hard *and* splittable.

We state the formal definitions. In these definitions you will see constants for probabilities. Notably $2/3$ *and* $1/100$*. These are arbitrary in that the properties we need are simply that* $1/2 < 2/3$ *and* $2/3$ *is much less than* $1 - 1/100$*. We later boost these probabilities and thus only care that there are constant gaps.*

Definition 33 (Average Case Indistinguishably Hard)**.** *For a decision or search problem* P *and instance size* n*, let* D *be the distribution drawing with probability* $1/2$ *from* $D_0(P, n)$ *and* $1/2$ *from* $D_1(P, n)$*.*

Let $val(I) = 0$ *if* I *is from the support of* D_0 *and let* $val(I) = 1$ *if* I *is from the support of* D_1*.*

P *is Average Case Indistinguishably Hard in time* $T(n)$ *(*$T(n)$*-ACIH) if* $T(n) = \Omega(n)$ *and for any* $\mathsf{PFT}_{T(n)}$ *algorithm* A

$$\Pr_{I \sim D}[A(I) = val(I)] \leq 2/3.$$

We also define a similar notion for search problems. Intuitively, it is hard to find a 'witness' for a problem with a solution, but we need to define what a witness is and how to verify a witness in the fine-grained world.

Definition 34 (Average Case Search Hard)**.** *For a search problem* P *and instance size* n*, let* $D_1 = D_1(P, n)$*.*

Let $wit(I)$ *denote an arbitrary witness of an instance* I *with at least one solution.*

P *is Average Case Search Hard in time* $T(n)$ *if* $T(n) = \Omega(n)$ *and*

- *there exists a* $\mathsf{PFT}_{T(n)}$ *algorithm* V *(a fine-grained verifier) such that* $V(I, wit(I)) = 1$ *if* I *has a solution and* $wit(I)$ *is a witness for it and* 0 *otherwise*
- *and for any* $\mathsf{PFT}_{T(n)}$ *algorithm* A

$$\Pr_{I \sim D_1}[A(I) = wit(I)] \leq 1/100.$$

Note that ACIH implies ACSH, but not the other way around. In fact, given difficulties in dealing with problems in the average case, getting search-to-decision reductions seems very difficult.

Our next definition describes a fine-grained version of a problem (or relation) being 'plantable' [98]. The definition of a plantable problem from Lindell's book states that a plantable NP-hard problem is hard if there exists a PPT sampling algorithm G*.* G *produces both a problem instance and a corresponding witness* (x, y)*, and over the randomness of* G*, any other PPT algorithm has a negligible chance of finding a witness for* x*.*

There are a couple of differences between our definition and the plantable definition from Lindell's book the [98]. First, we will of course have to put a fine-grained spin on it: our problem is solvable in time $T(n)$ *and so we will need to be secure against* $\mathsf{PFT}_{T(n)}$ *adversaries. Second, we will be focusing on a decision-version of our problems, as indicated by definition 33. Intuitively, our sampler (*$\mathsf{Generate}$*) will also take in a bit* b *to determine whether or not it produces an instance of the problem that has a solution or does not.*

Definition 35 (Plantable ($(G(n), \epsilon)$-Plantable)). *A $T(n)$-ACIH or $T(n)$-ACSH problem P is* plantable *in time $G(n)$ with error ϵ if there exists a randomized algorithm* $\mathsf{Generate}$ *that runs in time $G(n)$ such that on input n and $b \in \{0,1\}$, $\mathsf{Generate}(n, b)$ produces an instance of P of size n drawn from a distribution of total variation distance at most ϵ from $D_b(P, n)$.*

If it is a $T(n)-ACSH$ problem, then $\mathsf{Generate}(n, 1)$ also needs to output a witness $wit(I)$, in addition to an instance I.

We now introduce the List-Hard property. Intuitively, this property states that when given a list of length $\ell(n)$ of instances of P, it is almost as hard to determine if there exists one instance with a solution as it is to solve an instance of size $\ell(n) \cdot n$.

Definition 36 (Average Case List-hard ($(T(n), \ell(n), \delta_{LH})$-ACLH)). *A $T(n)$-ACIH or $T(n)$-ACSH problem P is* Average Case List Hard *in time $T(n)$ with list length $\ell(n)$ if $\ell(n) = n^{\Omega(1)}$, and for every $\mathsf{PFT}_{\ell(n)\cdot T(n)}$ algorithm A, given a list of $\ell(n)$ instances, $\mathbf{I} = I_1, I_2, \ldots, I_{\ell(n)}$, each of size n distributed as follows: $i \xleftarrow{\$} [\ell(n)]$ and $I_i \sim D_1(P, n)$ and for all $j \neq i$, $I_j \sim D_0(P, n)$;*

$$\Pr_{\mathbf{I}}[A(\mathbf{I}) = i] \leq \delta_{LH}.$$

It's worth noting that this definition is nontrivial only if $\ell(n) = n^{\Omega(1)}$. Otherwise $\ell(n)T(n) = \tilde{O}(T(n))$, since $\ell(n)$ would be sub-polynomial.

We now introduce the splittable property. Intuitively a splittable problem has a process in the average case to go from one instance I into a pair of average looking problems with the same number of solutions. We use the splittable property to enforce that a solution is shared between Alice and Bob, which becomes the basis of Alice and Bob's shared key (see Figure 4-2).

Definition 37 ((Generalized) Splittable). *A $T(n)$-ACIH problem P is generalized splittable with error ϵ, to the problem P' if there exists a $\mathsf{PFT}_{T(n)}$ algorithm* Split *and a constant m such that*

- *when given a P-instance $I \sim D_0(P, n)$, $\mathsf{Split}(I)$ produces a list of length m of pairs of instances $\{(I_1^1, I_2^1), \ldots, (I_1^m, I_2^m)\}$ where $\forall i \in [1, m]$ I_1^i, I_2^i are drawn from a distribution with total variation distance at most ϵ from $D_0(P', n) \times D_0(P', n)$.*
- *when given an instance of a problem $I \sim D_1(P, n)$, $\mathsf{Split}(I)$ produces a list of length m of pairs of instances $\{(I_1^1, I_2^1), \ldots, (I_1^m, I_2^m)\}$ where $\exists i \in [1, m]$ such that I_1^i, I_2^i are drawn from a distribution with total variation distance at most ϵ from $D_1(P', n) \times D_1(P', n)$.*

4.5.2 Concrete Hypothesis

Problem Descriptions *Two key problems within fine-grained complexity are the k-Sum problem and the Zero-k-Clique problem.*

Given k lists of n numbers $L_1, \ldots, L_k$, the k-Sum problem asks, are there $a_1 \in L_1, \ldots, a_k \in L_k$ so that $\sum_{j=1}^{k} a_j = 0$. The fastest known algorithms for k-Sum run in

$n^{\lceil k/2 \rceil - o(1)}$ *time, and this running time is conjectured to be optimal, in the worst case (see e.g. [112, 2, 128]).*

The Zero-k-Clique problem is, given a graph G on n vertices and integer edge weights, determine whether G contains k vertices that form a k-clique so that the sum of all the weights of the clique edges is 0*. The fastest known algorithms for this problem run in $n^{k-o(1)}$ time, and this is conjectured to be optimal in the worst case (see e.g. [13], [1], [97], [30]). As we will discuss later, Zero-k-Clique and k-Sum are related. In particular, it is known [130] that if* 3*-Sum requires $n^{2-o(1)}$ time, then Zero-*3*-Clique requires $n^{3-o(1)}$ time. Zero-*3*-Clique is potentially even harder than* 3*-Sum, as other problems such as All-Pairs Shortest Paths are known to be reducible to it, but not to* 3*-Sum.*

A folklore conjecture states that when the 3*-Sum instance is formed by drawing n integers uniformly at random from $\{-n^3, \ldots, n^3\}$ no PFT_{n^2} algorithm can solve* 3*-Sum on a constant fraction of the instances. This, and more related conjectures were explicitly formulated by Pettie [114].*

We propose a new hypothesis capturing the folklore intuition, while drawing some motivation from other average case hypotheses such as Planted Clique. For convenience, we consider the k-Sum and Zero-k-Clique problems modulo a number; this variant is at least as hard to solve as the original problems over the integers: we can reduce these original problems to their modular versions where the modulus is only k (for k-Sum) or $\binom{k}{2}$ (for Zero-k-Clique) times as large as the original range of the numbers.

We will discuss and motivate our hypotheses further in Section 4.6.

Definition 38. *An instance of the k-Sum problem over range R, k-Sum-R, consists of kn numbers in k lists $L_1, \ldots, L_k$. The numbers are chosen from the range $[0, R-1]$. A solution of a k-Sum-R instance is a set of k numbers $a_1 \in L_1, \ldots, a_k \in L_k$ such that their sum is zero mod R, $\sum_{i=1}^{k} a_i \equiv 0 \mod R$.*

We will also define the uniform distributions over k-Sum instances that have a certain number of solutions. We define two natural distributions over k-Sum-R instances.

Definition 39. *Define $D^{ksum}_{uniform}[R, n]$ be the distribution of instances obtained by picking each integer in the instance uniformly at random from the range $[0, R-1]$.*

Define $D^{ksum}_0[R, n] = D_0(\ k\text{-Sum-}R, n)$ to be the uniform distribution over k-Sum-R instances with no solutions. Similarly, let $D^{ksum}_1[R, n] = D_1(\ k\text{-Sum-}R, n)$ to be the uniform distribution over k-Sum-R instances with 1 *solution.*

The distribution $D_{ksum}[R, i, n]$ is the uniform distribution over k-Sum instances with n values chosen modulo R and where there are exactly i distinct solutions.

Let $D^{ksum}_0[R, n] = D_{ksum}[R, 0, n]$, and $D^{ksum}_1[R, n] = D_{ksum}[R, 1, n]$.

We now proceed to define the version of Zero-k-Clique that we will be using. In addition to working modulo an integer, we restrict our attention to k-partite graphs. In the worst case, the Zero-k-Clique on a general graph reduces to Zero-k-Clique on

a complete k-partite graph [2] *[8].*

Definition 40. *An instance of Zero-k-Clique-R consists of a k-partite graph with kn nodes and partitions $P_1, \ldots, P_k$. The k-partite graph is complete: there is an edge between a node $v \in P_i$ and a node $u \in P_j$ if and only if $i \neq j$. Thus, every instance has $\binom{k}{2}n^2$ edges. The weights of the edges come from the range $[0, R-1]$.*

A solution in a Zero-k-Clique-R instance is a set of k nodes $v_1 \in P_1, \ldots, v_k \in P_k$ such that the sum of all the weights on the $\binom{k}{2}$ edges in the k-clique formed by $v_1, \ldots, v_k$ is congruent to zero mod R: $\sum_{i \in [1,k]} \sum_{j \in [1,k] \text{ and } j \neq i} w(v_i, v_j) \equiv 0 \mod R$. A solution is also called a zero k-clique.

We now define natural distributions over Zero-k-Clique-R instances, similar to those we defined for k-Sum-R. We will additionally define the distributions of these instances in which a certain number of solutions appear.

Definition 41. *Define $D^{zkc}_{uniform}[R, n]$ to be the distribution of instances obtained by picking each integer edge weight in the instance uniformly at random from the range $[0, R-1]$.*

Define $D^{zkc}_0[R, n] = D_0(\text{Zero-k-Clique-R}, n)$ to be the uniform distribution over Zero-k-Clique-R instances with no solutions. Similarly, let $D^{zkc}_1[R, n] = D_1(\text{Zero-k-Clique-R}, n)$ to be the uniform distribution over Zero-k-Clique-R instances with 1 solution.

The distribution is $D_{zkc}[R, i, n]$ the uniform distribution over zero k-clique instances on kn nodes with weights chosen modulo R and where there are exactly i distinct zero k-cliques in the graph. Let $D^{zkc}_0[R, n] = D_{zkc}[R, 0, k]$ and $D^{zkc}_1[R, n] = D_{zkc}[R, 1, k]$.

Weak and Strong Hypotheses *The strongest hypothesis that one can make is that the average case version of a problem takes essentially the same time to solve as the worst case variant is hypothesized to take. The weakest but still useful hypothesis that one could make is that the average case version of a problem requires* super-linear *time. We formulate both such hypotheses and derive meaningful consequences from them.*

We state the weak versions in terms of decision problems and the strong version in terms of search problems. This is for convenience of presenting results. Our fine-grained one-way functions and fine-grained key exchanges can both be built using the search variants. We make these choices for clarity of presentation later on.

Definition 42 (Weak k-Sum-R Hypothesis)**.** *There exists some large enough constant c such that for all constants $c' > c$, distinguishing $D^{ksum}_0[c'R, n]$ and $D^{ksum}_1[c'R, n]$ is $n^{1+\delta}$-ACIH for some $\delta > 0$.*

Definition 43 (Weak Zero-k-Clique-R Hypothesis)**.** *There exists some large enough constant c such that for all constants $c' > c$, distinguishing $D^{zkc}_0[c'R, n]$ and $D^{zkc}_1[c'R, n]$ is $n^{2+\delta}$-ACIH for some $\delta > 0$.*

Notice that the Zero-k-Clique-R problem is of size $O(n^2)$.

[2] This reduction is done using color-coding ([8]), an example of this lemma exists in the paper "Tight Hardness for Shortest Cycles and Paths in Sparse Graphs" [97].

Definition 44 (Strong Zero-k-Clique-R Hypothesis for range n^{ck})**.** *For all $c > 1$, given an instance I drawn from the distribution $D_1^{zkc}[n^{ck}, n]$ where the witness (solution) is the single zero k-clique is formed by nodes $\{v_1, \ldots, v_k\}$, finding $\{v_1, \ldots, v_k\}$ is n^k-ACSH.*

Some may find the assumption with range n^k to be the most believable assumption. This is where the probability of a Zero-k-Clique existing at all is a constant.

Definition 45 (Random Edge Zero-k-Clique Hypothesis)**.** *Let $sol(I)$ be a function over instances of Zero-k-Clique problems where $sol(I) = 0$ if there are no zero k-cliques and $sol(I) = 1$ if there is at least one zero k-clique. Let $wit(I)$ be a zero k-clique in I, if one exists. Given an instance I drawn from the distribution $D_{uniform}^{zkc}[n^k, n]$ there is some large enough n such that for any PFT_{n^k} algorithm A*

$$\Pr_{I \sim D}[A(I) = wit(I) | sol(I) = 1] \leq 1/200.$$

Theorem 28. *Strong Zero-k-Clique-R Hypothesis for range $R = n^{ck}$ is implied by the Random Edge Random Edge Zero-k-Clique Hypothesis if $c > 1$ is a constant.*

The proof of this Theorem is in the appendix in Theorem 39. [3]

4.6 Our assumptions - background and justification

In this section, we justify making average-case hardness assumptions for k-SUM and Zero k-Clique — and why we do not for other fine-grained problems. We start with some background on these problems, and then justify why our hypotheses are believable.

4.6.1 Background for Fine-Grained Problems

Among the most popular hypotheses in fine-grained complexity is the one concerning the 3-Sum problem defined as follows: given three lists A, B and C of n numbers each from $\{-n^t, \ldots, n^t\}$ for large enough t, determine whether there are $a \in A, b \in B, c \in C$ with $a + b + c = 0$. There are multiple equivalent variants of the problem (see e.g. [59]).

The fastest 3-Sum algorithms run in $n^2(\log\log n)^{O(1)}/\log^2 n$ time (Baran, Demaine and Patrascu for integer inputs [18], and more recently Chan'18 for real inputs [37]). Since the 1990s, 3-Sum has been an important problem in computational geometry. Gajentaan and Overmars [59] formulated the hypothesis that 3-Sum requires quadratic time (nowadays this means $n^{2-o(1)}$ time on a word-RAM with $O(\log n)$ bit words), and showed via reductions that many geometry problems also require quadratic time under this hypothesis. Their work spawned many more within geometry. In recent years, many more consequences of this hypothesis have been derived, for a variety

[3] Thank you to Russell Impagliazzo for discussions related to the sizes of ranges R.

of non-geometric problems, such as sequence local alignment [1], triangle enumeration [112, 88], and others.

As shown by Vassilevska Williams and Williams [130], 3-Sum can be reduced to a graph problem, 0-Weight Triangle, so that if 3-Sum requires $n^{2-o(1)}$ time on inputs of size n, then 0-Weight Triangle requires $N^{3-o(1)}$ time in N-node graphs. In fact, Zero-Weight Triangle is potentially harder than 3-Sum, as one can also reduce to it the All-Pairs Shortest Paths (APSP) problem, which is widely believed to require essentially cubic time in the number of vertices. There is no known relationship (via reductions) between APSP and 3-Sum.

The Zero-Weight Triangle problem is as follows: given an n-node graph with edge weights in the range $\{-n^c, \ldots, n^c\}$ for large enough c, denoted by the function $w(\cdot,\cdot)$, are there three nodes p, q, r so that $w(p,q) + w(q,r) + w(r,p) = 0$? Zero-Weight Triangle is just Zero-3-Clique where the numbers are from a polynomial range.

An equivalent formulation assumes that the input graph is tripartite and complete (between partitions).

Both 3-Sum and Zero-Weight Triangle have generalizations for $k \geq 3$: k-Sumand Zero-Weight k-Clique, defined in the natural way: (1) given k lists of n numbers each from $\{-n^{ck}, \ldots, n^{ck}\}$ for large c, are there k numbers, one from each list, summing to 0? and (2) given a complete k-partite graph with edge weights from $\{-n^{kc}, \ldots, n^{kc}\}$ for large c, is there a k-clique with total weight sum 0?

4.6.2 Justifying the Hardness of Some Average-Case Fine-Grained Problems

The k-Sum problem is conjectured to require $n^{\lceil k/2 \rceil - o(1)}$ time (for large enough weights), and the Zero-Weight k-Clique problem is conjectured to require $n^{k-o(1)}$ time (for large enough weights), matching the best known algorithms for both problems (see [128]). Both of these conjectures have been used in fine-grained complexity to derive conditional lower bounds for other problems (e.g. [13], [1], [97], [30]).

It is tempting to conjecture average-case hardness for the key hard problems within fine-grained complexity: Orthogonal Vectors (OV), APSP, 3-Sum. However, it is known that APSP is not hard on average, for many natural distributions (see e.g. [113, 48]), and OV is likely not (quadratically) hard on average (see e.g. [84]).

On the other hand, it is a folklore belief that 3-Sum is actually hard on average. In particular, if one samples n integers uniformly at random from $\{-cn^3, \ldots, cn^3\}$ for constant c, the expected number of 3-Sums in the instance is $\Theta(1)$, and there is no known truly subquadratic time algorithm that can solve 3-Sum reliably on such instances. The conjecture that this is a hard distribution for 3-Sum was formulated for instance by Pettie [114].

The same folklore belief extends to k-Sum. Here a hard distribution seems to be to generate k lists uniformly from a large enough range $\{-cn^k, \ldots, cn^k\}$, so that the expected number of solutions is constant.

Due to the tight relationship between 3-Sum and Zero-Weight Triangle, one might also conjecture that uniformly generated instances of the latter problem are hard to

solve on average. In fact, if one goes through the reductions from the worst-case 3*-Sum problem to the worst-case Zero-Weight Triangle, via the* 3*-Sum Convolution problem [112, 131] starting from an instance of* 3*-Sum with numbers taken uniformly at random from a range, then one obtains a list of Zero-Weight Triangle instances that are essentially average-case. This is easier to see in the simpler but less efficient reduction in [131] which from a* 3*-Sum instance creates* $n^{1/3}$ *instances of (complete tripartite) Zero-Weight Triangle on* $O(n^{2/3})$ *nodes each and whose edge weights are exactly the numbers from the* 3*-Sum instance. Thus, at least for* $k = 3$*, average-case hardness for* 3*-Sum is strong evidence for the average-case hardness for Zero-Weight Triangle.*

In Appendix 39 we give a reduction between uniform instances of uniform Zero-Weight k*-Clique with range* $\Theta(n^k)$ *and instances of planted Zero-Weight* k*-Clique with large range. Working with instances of planted Zero-Weight* k*-Clique with large range is easier for our hardness constructions, so we use those in most of this paper.*

Justifying the Hardness of Distinguishing. *Now, our main assumptions consider distinguishing between the distributions* D_0 *and* D_1 *for* 3*-Sum and Zero-Weight Triangle. Here we take inspiration from the Planted Clique assumption from Complexity [70, 82, 90]. In Planted Clique, one first generates an Erdös-Renyi graph that is expected to not contain large cliques, and then with probability* $1/2$*, one plants a clique in a random location. Then the assertion is that no polynomial time algorithm can distinguish whether a clique was planted or not.*

We consider the same sort of process for Zero-k*-Clique. Imagine that we first generate a uniformly random instance that is expected to have no zero* k*-Cliques, by taking the edge weights uniformly at random from a large enough range, and then we plant a zero* k*-Clique with probability* $1/2$ *in a random location. Similarly to the Planted Clique assumption, but now in a fine-grained way, we can assume that distinguishing between the planted and the not-planted case is computationally difficult.*

Our actual hypothesis is that when one picks an instance that has no zero k*-Cliques at random with probability* $1/2$ *and picks one that has a zero* k*-Clique with probability* $1/2$*, then distinguishing these two cases is hard. As we show later, this hypothesis is essentially equivalent to the planted version (up to some slight difference between the underlying distributions).*

Similarly to Planted Clique, no known approach for Zero-k*-Clique seems to work in this average-case scenario, faster than essentially* n^k*, so it is natural to hypothesize that the problem is hard. We leave it as a tantalizing open problem to determine whether the problem is actually hard, either by reducing a popular worst-case hypothesis to it, or by providing a new algorithmic technique.*

4.7 Fine-Grained One-Way Functions

In this section, we give a construction of fine-grained OWFs (FGOWF) based on plantable $T(n)$*-ACIH problems. We first show that even though the probability of inversion may be constant (we call this "medium" fine-grained one-way), we can do some standard boosting in the same way weak OWFs can be transformed into strong*

OWFs in the traditional sense. Then, given such a plantable problem, we will prove that $\mathsf{Generate}(n, 1)$ *is a medium* $T(n)$*-FGOWF. Then, from this medium FGOWF, we can compile a strong FGOWF using this boosting trick. Then, since Zero-k-Clique is plantable (see Theorem 36), this implies that assuming Zero-k-Clique is hard yields fine-grained OWFs.*

Finally, we discuss the possibility of fine-grained hardcore bits and pseudorandom generators. It turns out that the standard Goldreich-Levin [64] approach to creating hardcore bits works in a similar fashion here, but requires some finessing; it will not work for all *fine-grained OWFs.*

We will be using $\tilde{O}(\cdot)$ *to suppress sub-polynomial factors of* n *(as opposed to only* $\lg(n)$ *factors).*

4.7.1 Weak and Strong OWFs in the Fine-Grained Setting

Traditional cryptography has notions of weak and strong OWFs. Weak OWFs can be inverted most of the time, but a polynomial-fraction of the time, they cannot be. These weak OWFs can be compiled into strong OWFs (showing that weak OWFs imply strong OWFs), where there is a negligible chance that the resulting strong OWF is invertible over the choice of inputs.

Here we will briefly define "medium" $T(n)$*-FGOWFs, and show how they can imply a "strong"* $T(n)$*-FGOWF, where "strong" refers to definition 28*

Definition 46. *A function* f *is a medium* $T(n)$*-FGOWF if there exists a sub-polynomial function* $Q(n)$ *such that for all* $\mathsf{PFT}_{T(n)}$ *adversaries* $\mathcal{A}$*,*

$$\Pr_{x \xleftarrow{\$} \{0,1\}^n} \left[\mathcal{A}(f(x)) \in f^{-1}(f(x))\right] \leq 1 - \frac{1}{Q(n)}.$$

Claim 14. *Medium* $T(n)$*-FGOWFs imply strong* $T(n)$*-FGOWFs for any polynomial* $T(n)$ *that is at least linear.*

Proof. The structure of this proof will follow Yao's original argument augmenting weak OWFs to strong ones. Intuitively, we are able to use this argument because sub-polynomial functions compose well with each other.

Assume for the sake of contradiction that no strong $T(n)$-FGOWF exists. Then, there exists some function $p'(n)$ such that $p'(n)$ is sub-polynomial and for all functions f computable in $T(n)^{1-\epsilon}$ time there exists a $PFT_{T(n)}$ adversary that can invert the function f with probability $1 - \frac{1}{p'(n)}$.

Let f be a medium $T(n)$-FGOWF, where the probability any $\mathsf{PFT}_{T(n)}$ adversary inverts it is $1 - \frac{1}{Q(n)}$. The basic idea will be to produce g that is just a concatenation of many fs, just as in the traditional cryptographic case.

For any positive-integer function $c(n)$, let $g(x_1||\ldots||x_{c(n)}) = f(x_1)||\ldots||f(x_{c(n)})$, where $||$ denotes concatenation. Let $c(n) = 4\left(Q(n)r(n)\right)^2$ (or the ceiling of $4\left(Q(n)r(n)\right)^2$, if not an integer). Where $r(n)$ is a subpolynomial function such that $r(n) \geq p'(c(n)\cdot n)$. Note that $p'(n)$ is subpolynomial, and that as a result some such function $r(n)$ exists. Specifically, setting $r(n) = p'(n^2)$ satisfies both criteria.

Now note that $c(n) \cdot n = \tilde{O}(n)$, and so $T(c(n) \cdot n) = \tilde{O}(T(n))$. Furthermore, g is a $T(c(n) \cdot n) = O(T(n))$-FGOWF since $c(n)$ is subpolynomial.

Now, for sake of contradiction, let $\mathcal{A}$ be a $\mathsf{PFT}_{T(n)}$ such that there exists a sub-polynomial function p' where

$$\Pr\Big[\mathcal{A}(g(x_1||\ldots||x_c)) \in g^{-1}(g(x_1||\ldots||x_c))\Big] \geq \frac{1}{p'(c(n)\cdot n)}.$$

Let $p(n) = p'(c(n) \cdot n)$. Because $c(n) \cdot n = O(n^2)$ and $p'(n)$ is sub-polynomial, $p'(c(n) \cdot n) = p(n)$ is also sub-polynomial. Therefore,

$$\Pr\Big[\mathcal{A}(g(x_1||\ldots||x_c)) \in g^{-1}(g(x_1||\ldots||x_c))\Big] \geq \frac{1}{p(n)}.$$

We will define a $\mathsf{PFT}_{T(n)}$ function $\mathcal{A}_0$ that makes a single call to $\mathcal{A}$: on input $y = f(x)$

1. Choose $i \xleftarrow{\$} [c(n)]$.
2. Let $z_i = y$
3. For all $j \in [c(n)]$, $j \neq i$, $x_j \xleftarrow{\$} \{0,1\}^n$ and $z_j = f(x_j)$.
4. Run $\mathcal{A}$ on $(z_1, \ldots, z_{c(n)})$ to get output $(x_1, \ldots, x_{c(n)})$ if $\mathcal{A}$ succeeds.
5. If $\mathcal{A}$ succeeded, output x_i.

Because all operations in $\mathcal{A}_0$ are either calling $\mathcal{A}$ (once) or take time $O(n \cdot c(n))$,[4] $\mathcal{A}_0$ is a $\mathsf{PFT}_{T(n)}$ algorithm. Now, we will let $\mathcal{B}$ be an algorithm calling $\mathcal{A}_0$ $d(n) = 4c(n)^2(n)p(n)Q(n)$ times, returning a valid inversion of $f(x)$ if $\mathcal{A}$ succeeded at least once.

We will call $x \in \{0,1\}^n$ 'good' if $\mathcal{A}_0$ inverts it with probability at least $\frac{1}{2c^2(n)p(n)}$; x is 'bad' otherwise. Notice that if x is good, then $\mathcal{B}$, which runs $\mathcal{A}$ many times, succeeds with high probability:

$$\Pr\left[\mathcal{B}(f(x)) \text{ fails} | x \text{ is good}\right] \leq \left(1 - \frac{1}{2c^2(n)p(n)}\right)^{d(n)} \sim e^{-2Q(n)} < \frac{1}{2Q(n)}.$$

We will show that there are at least $2^n(1 - \frac{1}{2p(n)})$ good elements.

Claim 15. *There are at least $2^n(1 - \frac{1}{2p(n)})$ good elements.*

Proof. For a contradiction, assume there are at least $2^n(\frac{1}{2p(n)})$ bad elements. We will end up contradicting the inversion probability of $\mathcal{A}$ (which is at least $1/p(n)$). For notation, let $\mathbf{x} = (x_1, \ldots, x_{c(n)}) \in \{0,1\}^{n \cdot c(n)}$, and $\mathbf{x}$ will be chosen uniformly at random over the input space.

$$\begin{aligned}\Pr_{\mathbf{x}}[\mathcal{A}(\mathbf{z} = g(\mathbf{x})) \text{ succeeds}] &= \Pr[\mathcal{A}(\mathbf{z}) \text{ succeeds} \wedge \exists \text{bad } x_j] \\ &\quad + \Pr\left[\mathcal{A}(\mathbf{z}) \text{ succeeds} \wedge \mathrm{x}_j \text{ good } \forall j \in [c(n)]\right]\end{aligned}$$

[4] It does not make much sense for $T(n)$ to be sublinear for our contexts

Now, for all $j \in [c(n)]$,

$$\begin{aligned}\Pr_{\mathbf{x}}[\mathcal{A}(\mathbf{z}) \text{ succeeds} \wedge x_j \text{ is bad}] &\leq \Pr_{\mathbf{x}}[\mathcal{A}(\mathbf{z}) \text{ succeeds} | x_j \text{ is bad}] \\ &\leq c(n) \Pr_{\mathbf{x}}[\mathcal{A}_0(f(x_j)) \text{ succeeds} | x_j \text{ is bad}] \\ &\leq \frac{c(n)}{2c^2(n)p(n)} = \frac{1}{2c(n)p(n)}\end{aligned}$$

So, if we just union bound over all j, we get

$$\Pr_{\mathbf{x}}[\mathcal{A}(\mathbf{z}) \text{ succeeds} \wedge \text{some } x_j \text{ are bad}] \leq \sum_{j=1}^{c(n)} \Pr_{\mathbf{x}}[\mathcal{A}_0(f(x_j)) \text{ succeeds} \wedge x_j \text{ is bad}] \leq \frac{1}{2p(n)}.$$

And one more quick upper bound yields

$$\begin{aligned}\Pr[\mathcal{A}(\mathbf{z}) \text{ succeeds} \wedge \text{all } x_j \text{ are good}] &\leq \Pr_{\mathbf{x}}[\text{all } x_j \text{ good}] \\ &< \left(1 - \frac{1}{2p(n)}\right)^{c(n)} \\ &\leq e^{-2(Q(n)r(n))^2/p(n)} \\ &\leq e^{-2(Q(n))^2 \cdot p(n)} < \frac{1}{2p(n)}.\end{aligned}$$

Finally, this yields the contradiction to the claim that there are at least $2^n(\frac{1}{2p(n)})$ bad elements:

$$\Pr_{\mathbf{x}}[\mathcal{A}(\mathbf{z}) \text{ succeeds}] < \frac{1}{p(n)}.$$

□ □

Note that $p(n) \geq Q(n)$ because $1/Q(n)$ is the maximum probability of inverting a single copy of $f(\cdot)$, where as $1/p(n)$ is assumed (for contradiction) to be the probability that a function inverts $c(n)$ copies of $f(\cdot)$ simultaneously. So, there are at most $2^n(\frac{1}{2Q(n)})$ bad elements.

Now that we know there is a high probability that we hit a good x, we can finish the rest of this proof.

$$\begin{aligned}\Pr_x[\mathcal{B}(f(x)) \text{ fails}] &= \Pr[\mathcal{B}(f(x)) \text{ fails} | x \text{ is good}] \Pr_x[x \text{ is good}] \\ &\quad + \Pr[\mathcal{B}(f(x)) \text{ fails} | x \text{ is bad}] \Pr_x[x \text{ is bad}] \\ &\leq \Pr[\mathcal{B}(f(x)) \text{ fails} | x \text{ is good}] \Pr_x[x \text{ is good}] + \Pr_x[x \text{ is bad}] \\ &\leq \frac{1}{2Q(n)}\left(1 - \frac{1}{2Q(n)}\right) + \frac{1}{2Q(n)} < \frac{1}{Q(n)}\end{aligned}$$

Thus, the chance that $\mathcal{B}$ actually has of inverting f is strictly greater than $1 - \frac{1}{Q(n)}$, contradicting the claim that f was medium-hard with respect to $Q(n)$. □ □

Weaker Fine-Grained OWFs. *Now, because we are in the fine-grained setting, we can talk about gaps. There is a notion of weak-OWFs in cryptography where we can say if there exists* any *polynomial such that we can invert with probability* $1 - 1/\text{poly}$*, we can construct strong OWFs. We want a similar notion for fine-grained OWFs. Here we can't just choose any polynomial — we have to choose a polynomial that respects the gap.*

Formally, for an $T(n)$*-FGOWF* f *that has* $\mathsf{PFT}_{T(n)}$ *adversaries inverting it with probability* $1 - 1/P(n)$ *for some* $P(n)$*, we can get that a* $\mathsf{PFT}_{T(n)}$ *adversary can invert* f *with probability* $(1 - 1/P(n))^{c(n)}$*. Now, as long as there exists* δ' *such that* $T(n)^{1-\delta}P(n) = T(n)^{1-\delta'}$*, there is still a gap (*$\delta' < \delta$*) even if we compute* f $P(n)$ *times to evaluate* f*. Therefore, we are able to get a strong fine-grained OWF from a weak one, as long as it's not* too *weak.*

4.7.2 Building Fine-Grained OWFs from Plantable Problems

Here we show that one can generate fine-grained one way functions from plantable problems. Recall the definition of Plantable states that there exists an algorithm $\mathsf{Generate}(n, b)$ *where when* $b = 0$*, an instance of the problem* without *a solution is generated, and when* $b = 1$*, an instance of a problem* with one *solution is generated with probability at least* $1 - \epsilon$*. This probabilistic element,* ϵ*, is actually a bound on the total variation distance of the distributions we are actually aiming to sample from:* $\mathsf{Generate}(n, 0)$ *and* $\mathsf{Generate}(n, 1)$ *have total variation distance at most* ϵ *from* $D_0(P, n)$ *and* $D_1(P, n)$ *respectively.*

Theorem 29. *If there exists a Plantable* $T(n)$*-ACIH problem where* $G(n)$ *is* $\mathsf{PFT}_{T(n)}$ *with error some constant* $\epsilon < 1/3$*, then* $T(n)$*-FGOWFs exist.* [5]

Proof. Let P be a Plantable $T(n)$-ACIH problem where $G(n) = T(n)^{1-\delta}$ for some constant $\delta > 0$. So, the (randomized) algorithm $\mathsf{Generate}(n, 1)$ is $\mathsf{PFT}_{T(n)}$ and outputs an instance I that has at least one solution — we write this as $\mathsf{Generate}(n, 1; r)$ when explicitly noting which randomness was used.

We want to show that being able to invert $\mathsf{Generate}(n, 1; r)$, over the distribution from r, in a fine-grained sense, as solving the ACIH problem P. Let ϵ be the upper bound on the total variation distance between $\mathsf{Generate}(n, 1)$ and $D_1(P, n)$, as per Definition 35.

For sake of contradiction, assume that no *medium* $T(n)$-FGOWF exist. So, we can invert $\mathsf{Generate}(n, 1)$ with any probability $1 - \frac{1}{Q(n)}$ for any sub-polynomial $Q(n)$. Let $\mathcal{A}$ be a $\mathsf{PFT}_{T(n)}$ algorithm that inverts $\mathsf{Generate}(n, 1)$ with probability $\gamma > 1 - \frac{1}{\log(n)}$ (note that $\log(n)$ is significant). We will show that this violates the assumption that P is a $T(n)$-ACIH problem.

We now construct a $\mathsf{PFT}_{T(n)}$ algorithm $\mathcal{B}$ that distinguishes between $I \sim D_0(P, n)$ and $I \sim D_1(P, n)$ with probability greater than $2/3$, violating the hardness assumption on P.

[5] We would like to thank Chris Brzuska and his reading group for finding a bug in the original version of this proof. To correct this bug, we added the word 'constant' to the theorem statement, removed our severe abuse of notation, and fixed the proof accordingly.

- Given I from distribution D, $\mathcal{B}$ gives I to $\mathcal{A}$.
- $\mathcal{A}$ outputs r.
- If $\mathsf{Generate}(n, 1; r) == I$, output 1. Otherwise, output 0.

We will now compute the probability that $\mathcal{B}$ distinguishes between inputs from D_1 and D_0. Recall that D is just sampling with D_0 with probability 1/2, and otherwise samples from D_1. For the sake of brevity let the notation $I \in D_0$ and $I \in D_1$ convey that I is in the support of D_0 and the support of D_1 respectively. We have

$$\begin{aligned}\Pr_{I\sim D}[\mathcal{B}(I) \text{ distinguishes } D_0 \text{ from } D_1] &= \Pr_{I\sim D_1}[\mathcal{B}(I) = 1] \cdot \Pr_{I\sim D}[I \in D_1] \\ &\quad + \Pr_{I\sim D_0}[\mathcal{B}(I) = 0] \cdot \Pr_{I\sim D}[I \in D_0] \\ &= \frac{1}{2}\Pr_{I\sim D_1}[\mathcal{B}(I) = 1] + \frac{1}{2}\Pr_{I\sim D_0}[\mathcal{B}(I) = 0].\end{aligned}$$

First, we note that $\Pr_{I\sim D_0}[\mathcal{B}(I) = 0] = 1$ because $\mathsf{Generate}(n, 1; r)$ is guaranteed to produce a witness for all randomness r. This means that any I sampled from D_0 is not in the image of $\mathsf{Generate}(n, 1)$, and therefore, $\mathcal{A}$ cannot produce a valid inverse.

Then, we use the fact that D_1 is close in total variation distance to $\mathsf{Generate}(n, 1)$ to show that $\Pr_{I\sim D_1}[\mathcal{B} = 1] \geq \gamma - 2\epsilon$. Let p_{G_1} be the pdf of $\mathsf{Generate}(n, 1)$ and p_{D_1} be the pdf of D_1. Let $\mathcal{I}$ be the set of all instances of the problem. Let $S = \{I \in \mathrm{Im}(\mathsf{Generate}(n, 1)) : \mathsf{Generate}(n, 1; \mathcal{A}(I)) = I\}$ be the set of instances produced by $\mathsf{Generate}$ that $\mathcal{A}$ can successfully invert. Recall that $\mathrm{TVD}(D_1, \mathsf{Generate}(n, 1)) \leq \epsilon$ means $\sum_{I\in\mathcal{I}} |p_D(I) - p_G(I)| \leq 2\epsilon$ by the definition of TVD.

$$\begin{aligned}2\epsilon &\geq \sum_{I\in\mathcal{I}} |p_{D_1}(I) - p_{G_1}(I)| \\ &\geq \sum_{I\in S} |p_{D_1}(I) - p_{G_1}(I)| \\ &\geq \sum_{I\in S} [p_{D_1}(I)] - \sum_{I\in S} [p_{G_1}(I)] .\end{aligned}$$

This implies $\sum_{I\in S} [p_{G_1}(I)] \geq \sum_{I\in S} [p_{D_1}(I)] - 2\epsilon$, and therefore $\Pr_{I\sim D_1}[\mathcal{B} = 1] \geq \gamma - 2\epsilon$.

Notice that since ϵ is constant and less than $\frac{1}{3}$, $\frac{1}{3} - \alpha = \epsilon$ for some constant $\alpha > 0$. Putting this together, we have that

$$\begin{aligned}\Pr_{I\sim D}[\mathcal{B}(I) \text{ distinguishes } D_0 \text{ from } D_1] &\geq \frac{1}{2} \cdot (\gamma - 2\epsilon) + \frac{1}{2} \\ &= \frac{\gamma}{2} - \epsilon + \frac{1}{2} \\ &= 1 - \frac{1}{2\log(n)} - \epsilon \\ &\geq 1 - \frac{1}{2\log(n)} - \left(\frac{1}{3} - \alpha\right) \\ &> \frac{2}{3}.\end{aligned}$$

Note that $\frac{1}{2\log(n)}$ is less (asymptotically) that any constant α, the sum of these terms is greater than $\frac{2}{3}$.

So, assuming P is $T(n)$-ACIH, i.e. no adversary has better than a constant chance less than 1 of being able to invert $\mathsf{Generate}(n, 1; r)$, then $\mathsf{Generate}$ is a medium $T(n)$-FGOWF. By Claim 14, this implies strong $T(n)$-FGOWFs exist. □ □

Note that k-Sum-R and Zero-k-Clique-R are plantable with error less than $1/3$ *the when* $R > 6n^k$ *by Theorem 36 and Theorem 35, these are both plantable and therefore can be used to build these fine-grained OWFs.*

4.7.3 Fine-Grained Hardcore Bits and Pseudorandom Generators

One way functions serve as the building block for a lot of symmetric encryption, and are (usually) implied by any other cryptographic primitive, from collision-resistant hash functions to symmetric-key encryption, to any flavor of public key encryption, and so on. The next step to building more cryptographic primitives with one-way functions is to see if we can use them to construct pseudorandom generators. While we do not yet have a construction of a fine-grained pseudorandom generator that can generate some sub-polynomial many pseudorandom bits[6]*, we take the first steps, showing how to get hardcore bits.*

Definition 47. *A function b is a fine-grained hardcore (FGHC) predicate for a* $T(n)$*-FGOWF if for all* $\mathsf{PFT}_{T(n)}$ *adversaries* $\mathcal{A}$*,*

$$\Pr_{x \xleftarrow{\$} \{0,1\}^n}[\mathcal{A}(f(x)) = b(x)] \leq \frac{1}{2} + \mathsf{insig}(n).$$

Recall that in traditional cryptography, any OWF implies the existence of another OWF with a hardcore bit with the Goldreich-Levin (GL) construction [64]. The bad news: the GL construction required a security reduction with $O(n)$ *evaluations of the one-way function. Given how we define problems to be* $T(n)$*-ACIH hard, this security reduction would not be* $\mathsf{PFT}_{T(n)}$*.*

Theorem 30 (Fine-Grained Goldreich-Levin)**.** *Let* f *be an* $T(n)$*-FG-OWF acting on strings* (x, y)*, where* $|x| = n$ *and* $|y| = Q(n)$ *for some subpolynomial* Q*, and assume there exists a* $\mathsf{PFT}_{T(n)}$ *algorithm* $\mathcal{L}$ *such that*

$$\Pr[\mathcal{L}(f(x, y), y) \in f^{-1}(f(x, y))] \geq \mathsf{sig}(n).$$

Then, the function $f' : (x, y, r) \mapsto f(x, y)||r$ *where* $|r| = |y|$ *has the hardcore bit* $y \cdot r$*.*

Proof. Here we just trace through the GL reduction and show that as long as $|y|$ is sub-polynomial, the reduction will go through.

[6]Note that due to the nature of being fine-grained, we cannot generate polynomially-many bits without additional assumptions

First, the size of y, $Q(n)$, cannot be any subpolynomial, it must be large enough so that it is as hard to guess y as it is to invert f (because guessing y yields a significant chance of inverting f). If f is $T(n)$-hard to invert, then the time it takes to randomly guess bits, $2^{Q(n)}$, must be at least $T(n)$. Since $T(n)$ is at least linear, we can assume $Q(n) \geq \log(n)$.

For a contradiction, assume that a $\mathsf{PFT}_{T(n)}$ adversary $\mathcal{A}$ has a significant advantage ϵ in determining $r \cdot y$ when given $f(x,y), r$. We will show this implies $\mathcal{B}$, a $\mathsf{PFT}_{T(n)}$ algorithm using $\mathcal{A}$, can invert f with significant probability.

$\mathcal{B}$ behaves as follows with parameter $m = 2Q(n)/\epsilon$ on input $x' = f(x,y)$:

- For every $i \in [Q(n)]$:
  1. Choose $\log(m)$ pairs $(b_1, r_1), \ldots, (b_{\log(m)}, r_{\log(m)}) \xleftarrow{\$} \{0,1\} \times \{0,1\}^{Q(n)}$.
  2. For every I in the powerset of $[\log(m)]$, let $b'_I = \sum_{j \in I} b_j \mod 2$.
  3. For every I in the powerset of $[\log(m)]$, let $r_I = \sum_{j \in I} r_j \mod 2$.
  4. For every I in the powerset of $[\log(m)]$,
     - Let $s_I \leftarrow e_i \oplus r_I$ where e_i is the i^{th} standard basis vector (e_i is all zeros except for one 1 in the i^{th} index).
     - Let $g_I \leftarrow b'_I \oplus \mathcal{A}(x'||s_I)$
  5. Let $z_i =$ the Majority bit over all $2^{\log(m)}$ bits g_I.
- Output $z = z_1, \ldots, z_{Q(n)}$.

First, consider the set $S = \left\{(x,y) | \Pr_r[\mathcal{A}(f(x,y)||r) = r \cdot y] \geq \frac{1}{2} + \frac{\epsilon}{2}\right\}$. A quick calculation yields $|S| > \epsilon \cdot 2^{n-1}$.

So, assume that our input $(x,y) \in S$. Now, assume that every pair we chose in step 1 has the property $b_i = r_i \cdot y$ (we correctly guessed the bit in question). This event occurs with probability $1/m$.

Next, notice that each pair of s_I, and s_J ($I \neq J$) are independent, and so the whole set is pairwise independent. So, if $(x,y) \in S$, $\mathcal{A}$ will return the correct bit given $f(x,y)||r_I$ at least an $\epsilon/2$-fraction of the time for independent r's. Because of the pairwise independence, a Chebyshev bound yields $\mathcal{A}$ will return the correct bit a majority of the time after the m queries (so $\mathsf{Majority}(\{g_I\})$ outputs the correct bit y_i) with probability at least $1 - \frac{1}{m(\epsilon/2)^2}$.

Finally, we put all of these pieces together to get

$$\begin{aligned}
\Pr[\mathcal{B}(f(x,y)) = y] &\geq \Pr[\mathcal{B}(f(x,y)) = y | (x,y) \in S] \cdot \frac{\epsilon}{2} \\
&= \frac{\epsilon}{2}\left(1 - \Pr[\exists i \text{ s.t. } y_i \neq z_i | (x,y) \in S]\right) \\
&\geq \frac{\epsilon}{2}\left(1 - Q(n)\Pr[y_i \neq z_i | (x,y) \in S]\right) \\
&\geq \frac{\epsilon}{2}\left(1 - Q(n)\Pr[y_i \neq z_i | (x,y) \in S \wedge \text{guessed all } b_i \text{ correctly}] \cdot \frac{1}{m}\right) \\
&\geq \frac{\epsilon}{2}\left(1 - \frac{Q(n)}{m} \cdot \frac{1}{m(\epsilon/2)^2}\right) \\
&= \frac{\epsilon}{2} - \frac{4Q(n)}{m^2\epsilon}
\end{aligned}$$

Recall we set $m = 2Q(n)/\epsilon$, and since ϵ is significant and Q is subpolynomial, m is also subpolynomial. Importantly, because $\mathcal{B}$ runs in $O(Q(n) \cdot mT(n)^{1-\delta})$, $\mathcal{B}$ is $\mathsf{PFT}_{T(n)}$.

Therefore, the probability that $\mathcal{B}$ succeeds in finding y_i (and hence inverting $f(x,y)$ with significant probability), with probability $\frac{\epsilon}{2} - \frac{4Q(n)\epsilon}{4Q^2(n)} \geq \frac{\epsilon}{2} - \frac{4\epsilon}{Q(n)}$. Recall that $Q(n)$ is at least linear in n, and so we can assume $Q(n) > 16$. This implies the probability $\mathcal{B}$ succeeds is $\frac{\epsilon}{4}$.

Because ϵ is significant, $\mathcal{B}$ breaks the fine-grained one-wayness of f. This is a contradiction. Therefore, ϵ must be insignificant. □ □

Hardcore bits from k-Sum and Zero-k-Clique

For both of these problems, planting a solution is exactly choosing some number of values (k for k-Sum, and the edge weights of a k-clique for Zero-k-Clique) and changing one of them so that the values now give a solution.

Corollary 8. *Assuming either the Weak k-Sum hypothesis or weak Zero-k-Clique hypothesis, there exist FGOWFs with fine-grained hardcore bits.*

Proof. This is straightforward due to the nature of planting for both of these hypotheses. Informally, planting for these problems is choosing a location within the given instance to put a solution. If an adversary learns where that solution is supposed to be, generating an instance without that specific solution is easy.

First, let's prove this for k-Sum. The reason k-Sum is plantable is because $\mathsf{Generate}(n, 1)$ chooses k indices at random in the k-Sum instance, and then changes the value one of them to make those k instances form a solution the k-Sum. This randomness requires specifying k instances out of kn, and an edge-weight. Let y be the $k\log(n)$ bits required to describe the k locations of the solution; y is part of the total randomness r used in $\mathsf{Generate}(n, 1)$. Without loss of generality, we can write $r = y||r'$. Let $f'(y||r', s) = \mathsf{Generate}(n, 1; y||r')||s$. Since $|y|$ is sub-polynomial, by Theorem 30, the bit $y \cdot s$ is hardcore for f'.

Now, let's make the same argument for Zero-k-Clique. As before, $\mathsf{Generate}(n, 1; r)$ can be written as $\mathsf{Generate}(n, 1; y||r')$ where y is the location of the zero k-clique generated. This location is just $k \cdot \log(n)$ bits; one coordinate from n for each of the k partitions in the graph. Therefore, we can define $f'(y||r', s) = \mathsf{Generate}(n, 1; y||r')||s$, which, by Theorem 30, has the hardcore bit $y \cdot s$. □ □

4.8 Fine-Grained Key Exchange

Now we will explain a construction for a key exchange *using general distributions. We will then specify the properties we need for problems to generate a secure key exchange. We will finally generate a key exchange using the strong Zero-k-Clique hypothesis. Sketches for most of proofs of these theorems are provided here, while full proofs can be found in Appendix* **??**.

Before doing this, we will define a class of problems as being Key Exchange Ready (KER).

Definition 48 (Key Exchange Ready (KER))**.** *A problem P is $\ell(n)$-KER with generate time $G(n)$, solve time $S(n)$ and lower bound solving time $T(n)$ if*

- *there is an algorithm which runs in $\tilde{\Theta}(S(n)))$ time that determines if an instance of P of size n has a solution or not,*
- *the problem is $(\ell(n), \delta_{LH})$-ACLH where $\delta_{LH} \leq \frac{1}{34}$,*
- *is Generalized Splittable with error $\leq 1/(128\ell(n))$ to the problem P' and,*
- *P' is plantable in time $G(n)$ with error $\leq 1/(128\ell(n))$.*
- *$\ell(n)T(n) \in \tilde{\omega}\left(\ell(n)G(n) + \sqrt{\ell(n)}S(n)\right)$, and*
- *there exists an n' such that for all $n \geq n'$, $\ell(n) \geq 2^{14}$.*

4.8.1 Description of a Weak Fine-Grained Interactive Key Exchange

The high level description of the key exchange is as follows. Alice and Bob each produce $\ell(n) - \sqrt{\ell(n)}$ instances using $\mathsf{Generate}(n, 0)$ and $\sqrt{\ell(n)}$ generate instances with $\mathsf{Generate}(n, 1)$. Alice then shuffles the list of $\ell(n)$ instances so that those with solutions are randomly distributed. Bob does the same thing (with his own private randomness). Call the set of indices that Alice chooses to plant solutions S_A and the set Bob picks S_B. The likely size of $S_A \cap S_B$ is 1. The index $S_A \cap S_B$ is the basis for the key.

Alice determines the index $S_A \cap S_B$ by brute forcing all problems at indices S_A that Bob published. Bob can brute force all problems at indices S_B that Alice published and learn the set $S_A \cap S_B$.

If after brute forcing for instances either Alice or Bob find a number of solutions not equal to 1 then they communicate this and repeat the procedure (using interaction). They only need to repeat a constant number of times.

More formally our key exchange does the following:

Construction 31 (Weak Fine-Grained Interactive Key Exchange)**.** *A fine-grained key exchange for exhanging a single bit key.*

- $\mathsf{Setup}(1^n)$*: output* $\text{MPK} = (n, \ell(n))$ *and* $\ell(n) > 2^{14}$.
- $\mathsf{KeyGen}(\text{MPK})$*: Alice and Bob both get parameters* (n, ℓ).
  - *Alice generates a random* $S_A \subset [\ell]$, $|S_A| = \sqrt{\ell}$. *She generates a list of instances* $\mathbf{I}_A = (I_A^1, \ldots, I_A^\ell)$ *where for all* $i \in S_A$, $I_i = \mathsf{Generate}(n, 1)$ *and for all* $i \notin S_A$, $I_A^i = \mathsf{Generate}(n, 0)$ *(using Alice's private randomness). Alice publishes* $\mathbf{I}_A$ *and a random vector* $\mathbf{v} \xleftarrow{\$} \{0,1\}^{\log \ell}$.
  - *Bob computes* $\mathbf{I}_B = (I_B^1, \ldots, I_B^\ell)$ *similarly: generating a random* $S_B \subset [\ell]$ *of size* $\sqrt{\ell}$ *and for every instance* $I_j \in \mathbf{I}_B$, *if* $j \in S_B$, $I_j = \mathsf{Generate}(n, 1)$ *and if* $j \notin S_B$, $I_j = \mathsf{Generate}(n, 0)$. *Bob publishes* $\mathbf{I}_B$.
- *Compute shared key: Alice receives* $\mathbf{I}_B$ *and Bob receives* $\mathbf{I}_A$.
  - *Alice computes what she believes is* $S_A \cap S_B$*: for every* $i \in S_A$, *she brute force checks if* I_B^i *has a solution or not. For each* i *that does, she records in list* L_A.
  - *Bob computes what he thinks to be* $S_B \cap S_A$*: for every* $j \in S_B$, *he checks if* I_A^j *has a solution. For each that does, he records it in* L_B.
- *Check: Alice takes her private list* L_A*: if* $|L_A| \neq 1$, *Alice publishes that the exchange failed. Bob does the same thing with his list* L_B*: if* $|L_B| \neq 1$, *Bob publishes that the exchange failed. If either Alice or Bob gave or recieved a failure, they both know, and go back to the* KeyGen *step.*

  If no failure occurred, then $|L_A| = |L_B| = 1$. *Alice interprets the index* $i \in L_A$ *as a vector and computes* $i \cdot \mathbf{v}$ *as her key. Bob uses the index in* $j \in L_B$ *and also computes* $j \cdot \mathbf{v}$. *With high probability,* $i = j$ *and so the keys are the same.*

4.8.2 Correctness and Soundness of the Key Exchange

We want to show that with high probability, once the key exchange succeeds, both Alice and Bob get the same shared index.

Lemma 26. *After running construction 31, Alice and Bob agree on a key* k *with probability at least* $1 - \frac{1}{10{,}000\ell e}$.

Sketch of Proof *We notice that the only way Alice and Bob fail to exchange a key is if they* both *generate a solution accidentally in each other's sets (that is Alice generates exactly one accidental solution in* S_B *and Bob in* S_A*), and* $S_A \cap S_B = \varnothing$.

All other 'failures' are detectable in this interactive case and simply require Alice and Bob to run the protocol again. So, we just bound the probability this happens, and since $\epsilon_{plant} \leq \frac{1}{100\sqrt{\ell}}$, we get the bound $1 - \frac{1}{10{,}000\ell e}$. The full proof can be found in Appendix 4.8.3. □

We next show that the key-exchange results in gaps in running time and success probability between Alice and Bob and Eve. Then, we will show that this scheme can be boosted in a fine-grained way to get larger probability gaps (a higher chance that Bob and Alice exchange a key and lower chance Eve gets it) while preserving the running time gaps.

First, we need to show that the time Alice and Bob take to compute a shared key is less (in a fine-grained sense) than the time it takes Eve, given the public transcript, to figure out the shared key. This includes the number of times we expect Alice and Bob to need to repeat the process before getting a usable key.

Time for Alice and Bob.

Lemma 27. *If a problem P is $\ell(n)$-KER with plant time $G(n)$, solve time $S(n)$ and lower bound $T(n)$ when $\ell(n) > 100$, then Alice and Bob take expected time $O(\ell G(n) + \sqrt{\ell} S(n))$ to run the key exchange.*

Sketch of Proof *It is easy to see that one iteration of the key exchange protocol requires $\ell G(n)$ time to generate the ℓ problems, and then $\sqrt{\ell} S(n)$ time to brute-force solve $\sqrt{\ell}$ instances of P. However, we need to prove that we only iterate this key-exchange a constant number of times. This part is a simple application of the birthday paradox, showing that we expect S_A and S_B to intersect in exactly one place with constant probability, and then applying the accuracy of our planting functionality (which succeeds with probability $1 - \epsilon_{plant}$). The full proof can be found in Appendix 4.8.4.* □

Time for Eve.

Lemma 28. *If a problem P is $\ell(n)$-KER with plant time $G(n)$, solve time $S(n)$ and lower bound $T(n)$ when $\ell(n) \geq 2^{14}$, then an eavesdropper Eve, when given the transcript $\mathbf{I}_T$, requires $\tilde{\Omega}(\ell(n)T(n))$ time to solve for the shared key with probability $\frac{1}{2} + \mathsf{sig}(n)$.*

Sketch of Proof *This is proved in two steps. First, if Eve can determine the shared key in time $\mathsf{PFT}_{\ell(n)T(n)}$ with advantage δ_{Eve}, then she can also figure out the index in $\mathsf{PFT}_{\ell(n)T(n)}$ time with probability $\delta_{Eve}/4$. Second, if Eve can compute the index with advantage $\delta_{Eve}/4$, we can use Eve to solve the list-version of P in $\mathsf{PFT}_{\ell(n)T(n)}$ with probability $\delta_{Eve}/16$, which is a contradiction to the list-hardness of our problem. This first part follows from a fine-grained Goldreich-Levin hardcore-bit theorem, Theorem 30.*

The second part, proving that once Eve has the index, then she can solve an instance of P, uses the fact that P is list-hard, generalized splittable, and plantable. Intuitively, since P is already list hard, we will start with a list of average problem instances $(I_1, \ldots, I_\ell)$, and our goal will be to have Eve tell us which instance (index)

has a solution. We apply the splittable property to this list to get lists of pairs of problems. For one of these lists of pairs, there will exist an index where both instances have solutions. These lists of pairs will almost *look like the transcript between Alice and Bob during the key exchange: if I had a solution then there should be one index such that both instances in a pair have a solution. Now, we just need to plant $\sqrt{\ell}-1$ solutions in the left instances and $\sqrt{\ell}-1$ on the right, and this will be indistinguishable from a transcript between Alice and Bob. If Eve can find the index of the pair with solutions, we can quickly check that she is right (because the instances inside the list are relatively small), and simply return that index.*

The full proof can be found in Appendix 4.8.4. □

Now, we can put all of these together to get a weak fine-grained key exchange. We will then boost it to be a strong fine-grained key exchange (see the Definition 30 for weak versus strong in this setting).

Theorem 32. *If a problem P is $\ell(n)$-KER with plant time $G(n)$, solve time $S(n)$ and lower bound $T(n)$ when $\ell(n) \geq 2^{14}$, then construction 31 is a $((\ell(n)T(n), \alpha, \gamma)$-*FG-KeyExchange*, with $\gamma \leq \frac{1}{10{,}000\ell(n)e}$ and $\alpha \leq \frac{1}{4}$.*

Proof. This is a simple combination of the correctness of the protocol, and the fact that an eavesdropper must take more time than the honest parties. We have that the $\Pr[b_A = b_B] \geq 1 - \frac{1}{10{,}000\ell e}$, implying $\gamma \leq \frac{1}{10{,}000\ell e}$ from Lemma 26. We have that Alice and Bob take time $O(\ell(n)G(n) + \sqrt{\ell(n)}S(n))$ and Eve must take time $\tilde{\Omega}(\ell(n)T(n))$ to get an advantage larger than $\frac{1}{4}$ by Lemmas 27 and 28. Because P is KER , $\ell(n)T(n) \in \tilde{\omega}\left(\ell(n)G(n) + \sqrt{\ell(n)}S(n)\right)$, implying there exists $\delta > 0$ so that $\ell(n)G(n) + \sqrt{\ell(n)}S(n) \in \tilde{O}(\ell(n)T(n)^{1-\delta})$. So, we have correctness, efficiency and security. □ □

Next, we are going to amplify the security of this key exchange using parallel repetition, drawing off of strategies from [56] and [23].

Theorem 33. *If a weak $(\ell(n)T(n), \alpha, \gamma)$-FG-KeyExchange exists where $\gamma = O\left(\frac{1}{n^c}\right)$ for some constant $c > 0$, but $\alpha = O(1)$, then a Strong $(\ell(n)T(n))$-FG-KeyExchange also exists.*

Proof. Using techniques from [23], we will phrase breaking this key exchange as an eavesdropper as an honest-verifier two-round parallel repetition game. The original game as a $\mathsf{PFT}_{\ell(n)T(n)}$ prover P and verifier V. V generates an honest transcript of the key exchange between Alice and Bob and sends this transcript to P. Note that the single-bit key sent in this protocol is uniformly distributed. P wins if P can output the key correctly. Now, because any eavesdropper running $\mathsf{PFT}_{\ell(n)T(n)}$ only has advantage α of determining the key, the prover P has probability at most $\frac{1}{2} + \alpha$ of winning this game. Let $\beta = \frac{1}{2} + \alpha$. By Theorem 4.1 of [23], if we instead have V generate m parallel repetitions (m independent transcripts of the key exchange), then the probability that P can find *all* of the keys in $\mathsf{PFT}_{\ell(n)T(n)}$ is less than $\frac{16}{1-\beta} \cdot e^{-m(1-\beta^2)/256}$ (which is larger than $\frac{32}{(1-\beta)} \cdot e^{-mc(1-\beta^2)/256}$).

Let $m = \frac{512}{1-\beta^2} \cdot \log(n)$, which is sub-polynomial in n because β is at least constant. and we get that the probability the prover succeeds in finding all m keys is at most $\beta' = O\left(\frac{1}{n^2}\right)$. Now, we need this to work while there is some error γ. Note that γ is at most $1/n^c$, so the probability we get an error in any of the m instances of the key exchange is $(1-\gamma)^{d \cdot \log(n)}$ for some constant d. Asymptotically, this is $e^{-\gamma d \log(n)} = n^{-\gamma d}$. This is where we required γ to be so small: because $\gamma = O(1/n^c)$, this probability of failure is $o(\sqrt{\gamma})$, which is still insignificant.

Now, we need to turn this m parallel-repetition back into a key exchange. We will first do that by employing our fine-grained Goldreich-Levin method: the weak key-exchange will be run m times in parallel and Alice will additionally send a uniformly random m-length binary vector, $\mathbf{r}$. The key will be the m keys, $\mathbf{k} = (k_1, \ldots, k_m)$ dot-producted with $\mathbf{r}$: $\mathbf{k} \cdot \mathbf{r}$. Because m is sub-polynomial in n and the Goldreich-Levin security reduction only requires $\tilde{O}(m^2)$ time, $\mathbf{k} \cdot \mathbf{r}$ is a fine-grained hard-core bit for the transcript. Therefore, an eavesdropper will have advantage at most $\frac{1}{2} + \mathsf{insig}(n)$ in determining the shared key. □ □

Remark 4. *It is not obvious how to amplify correctness* and *security of a fine-grained key exchange at the same time. If we have a weak* $(\ell(n)T(n), \alpha, \gamma)$*-FG-KeyExchange, where* $\alpha = \mathsf{insig}(n)$ *but* $\gamma = O(1)$*, then we can use a standard repetition error-correcting code to amplify* γ*. That is, we can run the key exchange* $\log^2(n)$ *times to get* $\log^2(n)$ *keys (most of which will agree between Alice and Bob), and to send a message with these keys, send that message* $\log^2(n)$ *times. With all but negligible probability, the decrypted message will agree with the sent message a majority of the time. Since with very high probability the adversary cannot recover any of the keys in* $\mathsf{PFT}_{\ell(n)T(n)}$ *time, this repetition scheme is still secure.*

As shown in Theorem 33, we can also amplify a key exchange that has constant correctness and polynomial soundness to one with $1 - \mathsf{insig}(n)$ *correctness and polynomial soundness. However, it is unclear how to amplify both at the same time in a fine-grained manner.*

Corollary 9. *If a problem* P *is* $\ell(n)$*-KER, then a Strong* $(\ell(n)T(n))$*-FG-KeyExchange exists.*

Proof. The probability of error in Construction 31 is at most $\frac{1}{10{,}000\ell(n)e}$, and $\ell(n) = n^{\Omega(1)}$ (due to the fact that $\ell(n)$ comes from the definition of list-hard, Definition 36. The probability a $\mathsf{PFT}_{\ell(n)T(n)}$ eavesdropper has of resolving the key is $\frac{1}{2} + \alpha$ where $\alpha \leq \frac{1}{4}$. This means our $\beta \leq \frac{3}{4} = O(1)$. Since the clauses in theorem 33 are met, a Strong $(\ell(n)T(n))$-FG-KeyExchange exists. □ □

Finally, using the fact that Alice and Bob do not use each other's messages to produce their own in Construction 31, we prove that we can remove all interaction through repetition and get a $T(n)$*-fine-grained public key cryptosystem.*

Lemma 29. *Construction 31 does not need interaction.*

Proof. There is a constant probability that the construction fails at each round. So, Alice and Bob will simply run the protocol $c \cdot \log(n)$ times in parallel and take the

key generated from the first successful exchange. There are two errors to keep track of: the chance that Alice and Bob's S_A and S_B do not intersect in exactly one spot and the probability that an instance was generated with a false-positive. Since ϵ_{plant} is so small ($O(1/n^{\Omega(1)})$), we do not need to worry about the false-positives (the chance of generating one is insignificant). So, the error we are concerned with is the chance that none of the $c\log(n)$ instances of the key exchange end up having S_A and S_B overlapping in exactly 1 entry. This happens with probability at most $\Pr[\text{no overlap } c\log(n) \text{ times}] = (\Pr[\text{no overlap once}])^{c\log n} \leq O(1/n^c)$, which is also insignificant. Therefore, the chance this key exchange fails is at most $1 - (\frac{c\log(n)}{10{,}000\ell e} + \frac{1}{n^c}) = 1 - \mathsf{insig}(n)$. □ □

Theorem 34. *If a problem P is $\ell(n)$-KER, then a $\ell(n)\cdot T(n)$-fine-grained public key cryptosystem exists.*

Proof. First, consider an amplified, non-interactive version of Construction 31 (combination of Corollary 9 and lemma 29): Alice and Bob run the protocol m times in parallel, where $m = \frac{512}{1-\beta^2}\cdot\log(n)$, and Alice sends an additional random binary vector $\mathbf{r} \in \{0,1\}^m$. The key they agree on is the dot-product between $\mathbf{r}$ and the vector of keys exchanged. This is a Strong $(\ell(n)T(n))$-FG- KeyExchange (see Corollary 9). We now define the three algorithms for a fine-grained public key cryptosystem.

- $\mathsf{KeyGen}(1^n)$: run Bob's half of the non-interactive protocol m times, generating m collections of $c\log(n)$ lists of $\ell(n)$ instances of P: $\{(\mathbf{I}_B^{(1,i)},\ldots,\mathbf{I}_B^{(c\log(n),i)})\}_{i\in[m]}$. Each list $\mathbf{I}_B^{(j,i)}$ has a random set $S_B^{(j,i)} \subset [\ell]$ where instances $I_k \in \mathbf{I}_B^{(j,i)}$ are from $\mathsf{Generate}(n,1)$ if $k \in S_B^{(j,i)}$ and from $\mathsf{Generate}(n,0)$ otherwise. The public key is $pk = \{(\mathbf{I}_B^{(1,i)},\ldots,\mathbf{I}_B^{(c\log(n),i)})\}_{i\in[m]}$ and the secret key is $sk = \{(S_B^{(1,i)},\ldots,S_B^{(c\log(n),i)})\}_{i\in m}$.

- $\mathsf{Encrypt}(pk, m \in \{0,1\})$: run Alice's half of the protocol m times, solving for the shared key, and then encrypting a message under that key. More formally, generate m lists of $c\log(n)$ sets $S_A^{(j,i)} \subset [\ell]$ such that $|S_A^{(j,i)}| = \sqrt{\ell}$. Then, generate lists of instances $\mathbf{I}_A^{(j,i)}$ for $i \in \{1,\ldots,m\}$ and $j \in \{1,\ldots,c\log(n)\}$, where for every instance $I_k^{(j,i)} \in \mathbf{I}_A^{(j,i)}$, $I_k^{(j,i)} = \mathsf{Generate}(n,1)$ if $k \in S_A^{(j,i)}$ and otherwise use $\mathsf{Generate}(n,0)$. Now, for each of these m instances, compute the shared key as in Construction the non-interactive version 31 (see lemma 29), to get a vector of keys $\mathbf{k} = (k_1,\ldots,k_m)$. If any of these exchanges fail, output $\perp$. Now, compute a random binary vector $\mathbf{r} \in \{0,1\}^m$ and let $key = \mathbf{k}\cdot\mathbf{r}$. Finally, let the ciphertext $c = \Big((\mathbf{I}_A^{(1)},\ldots,\mathbf{I}_A^{(m)}), \mathbf{r}, key \oplus m\Big)$.

- $\mathsf{Decrypt}(sk, c)$: Bob computes the shared key and decrypts the message. Formally, let $c = \Big((\mathbf{I}_A^{(1)},\ldots,\mathbf{I}_A^{(m)}), \mathbf{r}, c^*\Big)$. Bob computes the shared key just like Alice, using $(\mathbf{k}\cdot\mathbf{r}) \oplus c^* = m'$. Output the bit m'.

Now we will prove this is indeed a fine-grained public key cryptosystem. First, because the key exchange took Alice and Bob $\mathsf{PFT}_{\ell(n)T(n)}$ time, this scheme is efficient, requiring at most $(\ell(n)T(n))^{1-\delta}\cdot m\cdot(c\log n) = \tilde{O}(\ell(n)T(n)^{(1-\delta)})$ for constant $\delta > 0$.

Next, the scheme is correct. This comes directly from the fact that the key exchange succeeds with probability $1 - \mathsf{insig}(n)$.

Lastly, the scheme is secure. This is a simple reduction from the security game to an eavesdropper. For sake of contradiction, let Eve be a $\mathsf{PFT}_{\ell(n)T(n)}$ adversary that can win the CPA-security game as in Definition 31 with probability $\frac{1}{2}+\epsilon$ where $\epsilon = \mathsf{sig}(n)$. Eve can then directly win the key exchange wi†h the same advantage because the message the challenger gives to Eve is simply a transcript of a key exchange. □

Note that this encryption scheme can be used to send any sub-polynomial number of bits, just by running it in sequence sub-polynomially many times. We also want to note that the adversary's advantage cannot be any less than $\frac{1}{\mathrm{poly}(n)}$ since, due to the fine-grained nature of the scheme, the adversary can always solve the hard problem via guessing.

Corollary 10. *Given the strong Zero-k-Clique-R Hypothesis over range $R = \ell(n)^2 n^{2k}$, there exists a $(\ell(n)T(n), 1/4, \mathsf{insig}(n))$-FG-KeyExchange, where Alice and Bob can exchange a sub-polynomial-sized key in time $\tilde{O}\left(n^k\sqrt{\ell(n)} + n^2\ell(n)\right)$ for every polynomial $\ell(n) = n^{\Omega(1)}$.*

There also exists a $\ell(n)T(n)$-fine-grained public-key cryptosystem, where we can encrypt a sub-polynomial sized message in time $\tilde{O}\left(n^k\sqrt{\ell(n)} + n^2\ell(n)\right)$.

Both of these protocols are optimized when $\ell(n) = n^{2k-4}$.

Proof in Appendix **??**.

The Zero-3-Clique hypothesis (the Zero Triangle hypothesis) is generally better believed than the Zero-k-Clique hypothesis for larger k. Note that even with the strong Zero-3-Clique hypothesis we get a key exchange with a gap in the running times of Alice and Bob vs Eve. In this case, the gap is $t = 5/4 = 1.2$.

4.8.3 Proof of Correctness

First, we will prove Lemma 26. We have restated the lemma below.

Lemma 30. *After running construction 31, Alice and Bob agree on a key k with probability at least $1 - \frac{1}{10{,}000\ell e}$.*

Proof. Since we are allowing interaction, the only way Alice and Bob can fail is if one of Alice's $\mathsf{Generate}(n, 0)$ contains a solution that overlaps with S_B, one of Bob's $\mathsf{Generate}(n, 0)$ contains a solution that overlaps with S_A, *and* $S_A \cap S_B = \varnothing$.

First, let's compute $p_0 = \Pr[S_A \cap S_B = \varnothing]$. We have $p_0 = \prod_{i=0}^{\sqrt{\ell}} \left(\frac{\ell-\sqrt{\ell}-i}{\ell}\right)$, the chance that every time Bob chooses an element for S_B, he does not choose an element in S_A. Rearranging this expression, we have

$$p_0 = \prod_{i=0}^{\sqrt{\ell}-1}\left(\frac{\ell-\sqrt{\ell}-i}{\ell}\right) = \prod_{i=0}^{\sqrt{\ell}-1}(1-\frac{\sqrt{\ell}+i}{\ell}) \leq \prod_{i=0}^{\sqrt{\ell}-1}\left(1-\frac{1}{\sqrt{\ell}}\right) = \left(1-\frac{1}{\sqrt{\ell}}\right)^{\sqrt{\ell}} \approx \frac{1}{e}$$

Now, assuming that S_A and S_B do not intersect, we need to compute the probability that *both* Alice and Bob see an incorrectly generated instance (generated by

$\mathsf{Generate}(n, 0)$, but contains a solution). Let $\epsilon_{plant} \leq \frac{1}{100\ell}$ be the planting error. Since there is no overlap between S_A and S_B, these probabilities are independent. The probability that S_A overlaps is at most $\sqrt{\ell}\epsilon_{plant} \leq \frac{1}{100\sqrt{\ell}}$ via a union bound over all $\sqrt{\ell}$ instances corresponding to the indices in S_A. Therefore, the probability that this happens for both Alice and Bob is at most $\frac{1}{10{,}000\ell} = \left(\frac{\sqrt{\ell}}{10{,}000\ell}\right)^2$.

Thus, the probability that this event occurs is at most $\frac{1}{10{,}000\ell e}$, and it is the only way the protocol ends without Alice and Bob agreeing on a key.

Therefore, the probability Alice and Bob agree on a key at the end of the protocol is $1 - \frac{1}{10{,}000\ell e}$. □ □

4.8.4 Proof of Soundness

Here we will go over the full proofs that an adversary, Eve, must take more time that Alice and Bob to obtain the exchanged key. First, we upper bound the time Alice and Bob take. Then, we lower-bound the time Eve requires to break the key exchange assuming that we have a $\ell(n)$-KER problem P.

Lemma 31. *If a problem P is $\ell(n)$-KER with plant time $G(n)$, solve time $S(n)$ and lower bound $T(n)$ when $\ell(n) > 100$, then Alice and Bob take expected time $O(\ell G(n) + \sqrt{\ell} S(n))$ to run the key exchange based on P.*

Proof. First, we will compute a bound on the number of times Alice and Bob need to repeat the key exchange before they match on exactly one index. Alice and Bob repeat any time there isn't exactly one overlap between S_A and S_B or the key exchange fails, as described in the proof of Lemma 26. Since the probability of the bad event happening is small, $\leq 1/(10{,}000e\ell)$, we will ignore it. Instead, saying

$$\begin{aligned}
&\Pr[\text{Key Exchange Stops after this round}] \\
&\quad = \Pr[\text{bad event}] + \Pr[\text{Exactly one overlap}| \text{ no bad event}] \cdot \Pr[\text{no bad event}] \\
&\quad \geq \frac{1}{2} \Pr[\text{Exactly one overlap}| \text{ no bad event}] = \Pr[\text{ Exactly one overlap}]/2.
\end{aligned}$$

Computing the probability that there is exactly one overlap. Let p_0 and p_1 be the probability that there are zero overlaps and exactly 1 overlap respectively. First, using similar techniques as in the proof of Lemma 26, we show that $p_0 \geq \frac{1}{e^2}$

$$p_0 = \prod_{i=0}^{\sqrt{\ell}-1} (1 - \frac{\sqrt{\ell} + i}{\ell}) \geq \prod_{i=0}^{\sqrt{\ell}-1} \left(1 - \frac{2\sqrt{\ell}}{\ell}\right) = \left(1 - \frac{2}{\sqrt{\ell}}\right)^{\sqrt{\ell}} \approx \frac{1}{e^2}.$$

A combinatorial argument also tells us that $p_0 = \binom{\ell-\sqrt{\ell}}{\sqrt{\ell}} / \binom{\ell}{\sqrt{\ell}}$ since there are $\binom{\ell}{\sqrt{\ell}}$ possible ways to choose S_A independent of S_B, but if we want to ensure no overlap between S_A and S_B, we need to avoid the $\sqrt{\ell}$ locations in S_B, hence $\binom{\ell-\sqrt{\ell}}{\sqrt{\ell}}$ choices for S_A. Then, we have $p_1 = \sqrt{\ell} \cdot \binom{\ell-\sqrt{\ell}}{\sqrt{\ell}-1} / \binom{\ell}{\sqrt{\ell}}$ because there are $\sqrt{\ell}$ places to choose from to overlap S_A with S_B, and then we must avoid the $\sqrt{\ell} - 1$ locations in S_B for the rest of the $\sqrt{\ell}$ elements in S_A.

Now we will compute a bound on p_1 by first showing $\frac{p_1}{p_0} \geq 1$:

$$\begin{aligned}\frac{p_1}{p_0} &= \frac{\sqrt{\ell}\binom{\ell-\sqrt{\ell}}{\sqrt{\ell}-1}}{\binom{\ell}{\sqrt{\ell}}} \cdot \frac{\binom{\ell}{\sqrt{\ell}}}{\binom{\ell-\sqrt{\ell}}{\sqrt{\ell}}} \\ &= \frac{\sqrt{\ell}(\ell-\sqrt{\ell})!}{(\sqrt{\ell}-1)!(\ell-2\sqrt{\ell}+1)!} \cdot \frac{(\sqrt{\ell})!(\ell-2\sqrt{\ell})!}{(\ell-\sqrt{\ell})!} \\ &= \frac{(\sqrt{\ell})^2}{\ell-2\sqrt{\ell}+1} = \frac{\ell}{\ell-2\sqrt{\ell}+1} \geq 1\end{aligned}$$

Now, we have that $p_1 = \frac{p_1}{p_0} \cdot p_0 \geq 1 \cdot \frac{1}{e^2} \geq 1/10$.

Finally, putting this all together, the probability that Alice and Bob stop after a round of the protocol is at least $\frac{1}{20}$. And so, we expect Alice and Bob to stop after a constant number of rounds. Each round consists of calling $\mathsf{Generate}$ ℓ times and solving $\sqrt{\ell}$ instances; so, each round takes $\ell G(n) + \sqrt{\ell}S(n)$ time. Therefore, Alice and Bob take $O(\ell G(n) + \sqrt{\ell}S(n))$. □ □

Now, proving a lower bound on Eve's time.

Lemma 32. *If a problem P is $\ell(n)$-KER with plant time $G(n)$, solve time $S(n)$ and lower bound $T(n)$ when $\ell(n) \geq 2^{14}$, then an eavesdropper Eve, when given the transcript $\mathbf{I}_T$, requires $\tilde{\Omega}(\ell(n)T(n))$ to solve for the shared key.*

Proof. This proof requires two steps: first, if Eve can figure out the shared key in time $\mathsf{PFT}_{\ell(n)T(n)}$ time with advantage δ_{Eve}, then she can also figure out the index in $\mathsf{PFT}_{\ell(n)T(n)}$ time with probability $\delta_{Eve}/4$. Then, if Eve can compute the index with advantage $\delta_{Eve}/4$, we can use Eve to solve the list-version of P in $\mathsf{PFT}_{\ell(n)T(n)}$ with probability $\delta_{Eve}/16$, which is a contradiction to the list-hardness of our problem.

Finding a bit finds the index. This is just the Goldreich-Levin (GL) trick used in classical cryptography to convert OWFs to OWFs with a hardcore bit. We have to be careful in this scenario since the security reduction for GL requires polynomial overhead ($O(N^2)$). However, this is only because we are trying to find N bits based off of linear combinations of those bits. If instead we were trying to find $\operatorname{poly}\log N$ bits, we would only require $\operatorname{poly}\log N$ time to do so with this trick. $i \in \ell(n)$ is an index, so $|i| = \log(\ell(n))$. Because $\ell(n)$ is polynomial in n, $|i|$ is polynomial in the log of n, therefore, using the same techniques as used in the proof of Theorem 30, being able to determine $i \oplus r$ with δ advantage allows us to determine i in the same amount of time, with probability $\delta/4$.

Finding the index solves P Now, let $\mathbf{I} = (I_1, \ldots, I_\ell)$ be an instance of the list problem for P: for a random index i, $I_i \leftarrow D_1$, and for all other $j \neq i, I_j \leftarrow D_0$. Because P is generalized splittable, we can take every I_i and turn it into a list of m instances. With probability $1 - \ell\epsilon_{split}$, we turn $\mathbf{I}$ to m different instances: for every $c \in [m]$, $\mathbf{I}^{(c)} = ((I_1^{(1,c)}, I_1^{(2,c)}), \ldots (I_\ell^{(1,c)}, I_\ell^{(2,c)}))$. For all c and $j \neq i$, $(I_j^{(1,c)}, I_j^{(2,c)}) \sim D_0 \times D_0$, and for at least one $c^* \in [m]$, $(I_i^{(1,c^*)}, I_i^{(2,c^*)}) \sim D_1 \times D_1$. Because P is

plantable, for $\sqrt{\ell}-1$ random coordinates $h \in [\ell]$, for all $c \in [m]$, we will change $I_h^{(1,c)}$ to $I_h^{\prime(1,c)} \sim \mathsf{Generate}(n,1)$, and for $\sqrt{\ell}-1$ random coordinates $g \in [\ell]$, disjoint from all h's, for every $c \in [m]$, we will similarly plant solutions in the second list, changing $I_g^{(2,c)}$ to $I_g^{\prime(2,c)} \sim \mathsf{Generate}(n,1)$.

Note that there are ℓ instances and Eve returns a single index. We can verify the correctness by brute forcing a single instance in the list instance. When ℓ is polynomial in n then the time to brute force is polynomially smaller than the time required to solve the list instance. We will need to brute force m of these instances (one for each of the m produced pairs of lists). When $\ell/m = n^{-\Omega(1)}$, the total time for all the brute forces is polynomially smaller than the time required for solving a single list instance. This is how we deal with the "dummy" instances produced with by the splittable construction.

Now, notice that we have changed the list version of the problem into m different lists of pairs of instances, $\left\{\left(\mathbf{I}_1^{(c)}, \mathbf{I}_2^{(c)}\right)\right\}_{c\in[m]}$, and there exists a c^* such that the c^*'th list is distributed, with probability $O(1-1/\sqrt{\ell})$, indistinguishably to the transcript of a successful key exchange between Alice and Bob. We planted $\sqrt{\ell}-1$ solutions into random indices, and as long as we avoided the index with the solution (which happens with probability $1-\frac{2}{\sqrt{\ell}}$), the rest of the pairs will be of the form $D_0 \times D_0$ with exactly one coordinate of overlapping instances with solutions. That coordinate will be the same as the index in the list problem with the solution.

So, since we are assuming Eve can run in $\mathsf{PFT}_{\ell(n)T(n)}$ time and we can create instances that look like key-exchange transcripts from list-problems, we can run Eve on each of these m different list-pair problems, and as long as she answers correctly for the c^* instance, we can solve our original problem in time $O((\ell(n)T(n))^{1-\delta})$ for $\delta > 0$. This is a contradiction to the hardness of the list problem, meaning Eve's time is bounded by $\Omega(\ell(n)T(n))$.

Analyzing the error in this case, when the key exchange succeeds, the total variation distance between an instance of the list problem being split and the original key-exchange transcript is bounded above by the following two sides:

- For the c^* that splits the D_1 instance of the list into one sampled from $D_1 \times D_1$, this succeeds with probability $1-\epsilon_{split}\cdot\ell$.

- Given that we successfully split, the distance between the generated pairs of lists *after* we plant $\sqrt{\ell}-1$ instances with a solution between this and the idealized list of $(D_b, D_{b'})$ instances with one (D_1, D_1), $\sqrt{\ell}-1$ of the form (D_1, D_0) and $\sqrt{\ell}-1$ of the form (D_0, D_1) is at most $\frac{2}{\sqrt{\ell}} + (1-\frac{2}{\sqrt{\ell}})(\sqrt{\ell}\cdot\epsilon_{plant}) \leq \frac{2}{\sqrt{\ell}} + \sqrt{\ell}\epsilon_{plant}$.

- For the generated instances generated in a successful key exchange transcript, the error between this and the idealized list-pairs (described above) is at most $\ell \cdot \epsilon_{plant}$.

- Recall that $\epsilon_{plant}, \epsilon_{split} \leq \frac{1}{100\ell}$ and that $\ell \geq 2^{14}$. So, combined, the key-exchange transcript distribution and splitting the list-hard problem distribution are in-

distinguishable with probability at most

$$\begin{aligned}
&1 - (\epsilon_{split}\ell + \frac{2}{\sqrt{\ell}} + \sqrt{\ell}\epsilon_{plant} + \ell\epsilon_{plant}) \\
&= 1 - (\ell(\epsilon_{split} + \epsilon_{plant}) + \sqrt{\ell}\epsilon_{plant} + \frac{2}{\sqrt{\ell}}) \\
&\geq 1 - (\ell(\frac{2}{128\ell}) + \frac{1}{128\sqrt{\ell}} + \frac{2}{\sqrt{\ell}}) \\
&\geq 1 - (\frac{2}{128} + \frac{2}{128} + \frac{1}{128^2}) = 1 - \frac{1}{32} - \frac{1}{2^{14}} \\
&> 1 - \frac{1}{31}
\end{aligned}$$

Therefore, the total variation distance between key-exchange transcripts and the transformed ACLH instances is at most $\frac{1}{31}$.

Now, recall that if we have a $\mathsf{PFT}_{\ell(n)T(n)}$ algorithm E that resolves the single-bit key with advantage δ, then there exists a $\mathsf{PFT}_{\ell(n)T(n)}$ algorithm E^* that resolves the index of the key exchange transcript with probability $\delta/4$. Let $Transf$ be the algorithm that transforms an ACLH instance $\mathbf{I}$ to the key-exchange transcript (with TVD from a successful key-exchange transcript of $\frac{1}{34}$) Therefore, the probability that we fool Eve into solving our ACLH problem is

$$\Pr[E^*(Transf(\mathbf{I})) = i] \geq \delta/4 - \frac{1}{31} \geq \frac{1}{16} - \frac{1}{31} > \frac{1}{34}$$

Now, since the ACLH problem P allows for $\mathsf{PFT}_{\ell(n)T(n)}$ adversaries to have advantage at most $\frac{1}{34}$, this is a contradiction. Therefore, there does not exist a $\mathsf{PFT}_{\ell(n)T(n)}$ eavesdropping adversary that can resolve the single bit key with advantage $\frac{1}{4}$ (so resolving the key with probability $1/2 + 1/4 = 3/4$). □ □

We note that the range, $R \approx n^{6k}$ in the above corollary may be considered to be "too large" if you believe the hardness in the problem comes from a range where were are expected to get one solution with probability 1/2 ($R = O(n^k)$). So, in the next corollary, we address that problem, getting the key exchange using this much smaller range.

We will now provide the proof for Corollary 10. We will repeat The text of the Corollary here.

Corollary 10. Given the strong Zero-k-Clique-R Hypothesis over range $R = \ell(n)^2 n^{2k}$, there exists a
$(\ell(n)T(n), 1/4, \mathsf{insig}(n))$-$\mathsf{FG\text{-}KeyExchange}$, where Alice and Bob can exchange a sub-polynomial-sized key in time $\tilde{O}\left(n^k\sqrt{\ell(n)} + n^2\ell(n)\right)$ for every polynomial $\ell(n) = n^{\Omega(1)}$.

There also exists a $\ell(n)T(n)$-fine-grained public-key cryptosystem, where we can encrypt a sub-polynomial sized message in time $\tilde{O}\left(n^k\sqrt{\ell(n)} + n^2\ell(n)\right)$.

Both of these protocols are optimized when $\ell(n) = n^{2k-4}$.

Proof. This comes from the fact that strong Zero-k-Clique-R Hypothesis implies that Zero-k-Clique is a KER problem by Theorem 38, Theorem 37, and Theorem 36. So we can use construction 31 to get the key-exchange by Theorem 32 and Claim 14.

The optimization comes from minimizing $\tilde{O}\left(n^k\sqrt{\ell(n)} + n^2\ell(n)\right)$, which is simply to set $n^k\sqrt{\ell(n)} = n^2\ell(n)$. This results in $\ell(n) = n^{2k-4}$.

The gap between honest parties and dishonest parties is computed as follows. Honest parties take $H(n) = \tilde{O}(\ell(n)n^2) = \tilde{O}(n^{2k-2})$. Dishonest parties take $E(n) = \tilde{O}(\ell(n)n^k) = \tilde{O}(n^{3k-4})$. We have that $E(n) = H(n)^t$ where $t = \frac{3k-4}{2k-2}$, which approaches 1.5 as $k \to \infty$. So, we have close to a 1.5 gap between honest parties and dishonest ones as long as we assume $T(n) = n^k$. □ □

Corollary 11. *Given the strong Zero-k-Clique-R Hypothesis over range $R = n^k$, where $\ell(n)$ is polynomial, there exists a $(\ell(n)T(n), 1/4, \mathsf{insig}(n))$-FG-KeyExchange, where Alice and Bob can exchange a sub-polynomial-sized key in time $\tilde{O}\left(n^k\sqrt{\ell(n)} + n^2\ell(n)\right)$ for every polynomial $\ell(n) = n^{\Omega(1)}$.*

There also exists a $\ell(n)T(n)$-fine-grained public-key cryptosystem, where we can encrypt a sub-polynomial sized message in time $\tilde{O}\left(n^k\sqrt{\ell(n)} + n^2\ell(n)\right)$.

Both of these protocols are optimized when $\ell(n) = n^{2k-4}$.

Proof. Using Corollary 10 and Theorem 39 we can use the hardness of strong Zero-k-Clique-R Hypothesis over range $R = n^k$ to show hardness for strong Zero-k-Clique-R Hypothesis over range $R = \ell(n)^2 n^{2k}$. □ □

4.9 Properties of k-Sum and Zero-k-Clique Hypotheses

In this section, we will prove the properties that k-Sum and Zero-k-Clique have that will make them useful in constructing fine-grained OWFs and our fine-grained key exchange.

4.9.1 k-Sum is Plantable from a Weak Hypothesis

Here we will show that by assuming the Weak k-Sum hypothesis (see definition 42), we get that k-Sum is plantable and $n^{2+\delta}$-ACIH. The proof is relatively straightforward: just show that planting a solution in a random k-Sum-R instance is easy while making sure that the distributions are close to what you expect.

Theorem 35. *Assuming the weak k-Sum-R hypothesis, k-Sum-R is plantable with error $\leq 2n^k/R$ in $O(n)$ time.*

Proof. First, we will define $\mathsf{Generate}(n, b)$:

- $b = 0$: choose all kn entries uniformly at random from $[0, R-1]$, taking time $O(n)$.

- $b = 1$: choose all kn entries uniformly at random from $[0, R-1]$, then choose values $v_1, \ldots, v_k$, each v_i at random from partition P_i, and choose $i \xleftarrow{\$} [k]$. Set $v_i = -\sum_{j \neq i} v_j \mod R$. This takes time $O(n)$.

We need to show that $\mathsf{Generate}(n, 0)$ is ϵ-close to D_0 and $\mathsf{Generate}(n, 1)$ is ϵ-close to D_1.

First, we note that $\mathsf{Generate}(n, 0)$ has the following property: $\Pr_{I \sim \mathsf{Generate}(n,0)}[I = I' | I \text{ has no solutions}] = \Pr_{I \sim D_0}[I = I']$. This is because $\mathsf{Generate}(n, 0)$ samples uniformly over the support of D_0. So, the total variation distance between $\mathsf{Generate}(n, 0)$ and D_0 is the probability $\mathsf{Generate}(n, 0)$ samples outside of the support of D_0, that is, the probability $\mathsf{Generate}(n, 0)$ samples an I with a value 1 or greater. Let TVD denote Total Variation Distance between two distributions. Now, a union bound gives us

$$\begin{aligned}
\mathrm{TVD}(\mathsf{Generate}(n, 0), D_0) &= \Pr_{I \sim \mathsf{Generate}(n,0)}[I \text{ has at least 1 solution}] \\
&\leq \sum_{\text{all } n^k \text{ sums } \mathbf{s} \in [n]^k} \Pr_{I \sim \mathsf{Generate}(n,0)}[\mathbf{s} \text{ is a } k\text{-Sum}] \\
&= \frac{n^k}{R}.
\end{aligned}$$

Now, to show that $\mathsf{Generate}(n, 1)$ is ϵ-close to D_1, we will use the fact that total-variation distance (TVD) is a metric and the triangle inequality. Let $\mathsf{Generate}(n, 0)+$ Plant and D_0+ Plant denote sampling from the first distribution and planting a k-Sum solution at random (so $\mathsf{Generate}(n, 0)+$ Plant $= \mathsf{Generate}(n, 1)$). We have that

$$\begin{aligned}
\mathrm{TVD}(\mathsf{Generate}(n, 1), D_1) \leq\ & \mathrm{TVD}(\mathsf{Generate}(n, 0) + \text{Plant}, D_0 + \text{Plant}) \\
& + \mathrm{TVD}(D_0 + \text{Plant}, D_1).
\end{aligned}$$

The distance $\mathsf{Generate}(n, 0)+$ Plant from D_0+ Plant is equal to the distance from $\mathsf{Generate}(n, 0)$ and D_0, since the planting does not change between distributions. As previously shown, this distance is at most $\frac{n^k}{R}$. The distance from D_0+ Plant and D_1 is just the chance that we introduce more than one clique by planting. We are only changing one value in the D_0 instance, v_i. There are $n^{k-1} - 1 \leq n^{k-1}$ possible sums involving v_i, so the chance that we accidentally introduce an unintended k-Sum solution is at most $\frac{n^{k-1}}{R}$. Therefore,

$$\mathrm{TVD}(\mathsf{Generate}(n, 1), D_1) \leq \frac{n^k}{R} + \frac{n^{k-1}}{R} < \frac{2n^k}{R}$$

□ □

Note that when $R > 6n^k$, $\mathsf{Generate}(n, 1)$ has total variation distance $< 1/3$ from $D_1(k{-}SUM{-}R, n)$.

4.9.2 Zero-k-Clique is also Plantable from Weak or Strong Hypotheses

The proof in this section mirrors of the proof that k-Sum-R is plantable. Note that the size of a k-Clique instance is $O(n^2)$, and so the fact that this requires $O(n^2)$ time

is just that it is linear in the input size. Here we will just list what the Generate *functionality is:*

- $\mathsf{Generate}(n, 0)$ *outputs a complete* k*-partite graph with* n *nodes in each partition, and edge weights drawn uniformly from* $\mathbb{Z}_R$*. This takes* $O(n^2)$ *time.*

- $\mathsf{Generate}(n, 1)$ *starts with* $\mathsf{Generate}(n, 0)$*, and then plants a clique by choosing a node from each partition,* $v_1 \in P_1, \ldots, v_k \in P_k$*, choosing an* $i \neq j \xleftarrow{\$} [k]$*, and setting the weight* $w(v_i, v_j) = -\sum_{(i',j') \neq (i,j)} w(v_{i'}, v_{j'}) \mod R$*. This also takes* $O(n^2)$ *time.*
  If assuming the strong hypothesis (search problem), we can also output a witness, $(v_1, \ldots, v_k)$*, of size* $O(\log n)$*.*

Unfortunately for it seems difficult to show that k*-Sum is average-case list-hard or splittable. However, we will show that if we assume that Zero-*k*-Clique is only* search *hard (a strictly weaker assumption than being indistinguishably hard), we can get the plantable, list-hard, and splittable properties — the caveat is that we need to assume that Zero-*k*-Clique requires* $\tilde{\Omega}(n^k)$ *time to solve (not just super-linear in time).*

Before proving the theorem, we need a couple of helper lemmas to characterize the total variation distance, etc. These lemmas will be useful later on as well.

Lemma 33. *The distribution* $D_0^{zkc}[R, n]$ *has total variation distance* $\leq n^k/R$ *from the distribution of instances drawn from* $\mathsf{Generate}(n, 0)$*.*

Proof. $D_0^{zkc}[R, n]$ is uniform over all instances of size n where there are no solutions. $\mathsf{Generate}(n, 0)$ is uniform over all instances of size n.

Let D be the distribution of instances in $\mathsf{Generate}(n, 0)$ which are in the support of $D_0^{zkc}[R, n]$. Because both $\mathsf{Generate}(n, 0)$ and $D_0^{zkc}[R, n]$ are uniform over the support of $D_0^{zkc}[R, n]$, $D = D_0^{zkc}[R, n]$.

So the total variation distance between $D_0^{zkc}[R, n]$ and $\mathsf{Generate}(n, 0)$ is just

$$Pr_{I \sim \mathsf{Generate}(n,0)}[I \notin \text{ the support of } D_0^{zkc}[R, n]].$$

The expected number of zero k-cliques is n^k/R, every set of k nodes has a chance of $1/R$ of being a zero k-clique. Thus, the probability that an instance has a non-zero number of solutions is $\leq n^k/R$. So, the total variation distance is $\leq n^k/R$. □

Lemma 34. *The distribution* $D_1^{zkc}[R, n]$ *has total variation distance* $\leq n^k/R + n^{k-2}/R$ *from the distribution of* $\mathsf{Generate}(n, 1)$*.*

Proof. We want to first show that $\mathsf{Generate}(n, 1)$ is uniform over the support of $D_1^{zkc}[R, n]$. Consider an instance I in the support of $D_1^{zkc}[R, n]$. Let $S(I) = a_1, \ldots, a_k$ be the set of k nodes in which there is a zero k-clique. $Pr_{I' \sim \mathsf{Generate}(n,1)}[I' = I]$ is given by the chance that

- the nodes chosen in I' $(a'_1, \ldots, a'_k)$ to plant a clique are the same as those in $S(I)$,

- the edges in the clique have the same weights in I' and I and,

- all edges outside the clique have the same weight in I' and I.

$$Pr_{I' \sim \mathsf{Generate}(n,1)}[I' = I] = \left(n^{-k}\right)\left(R^{-\binom{k}{2}-1}\right)\left(R^{-\binom{k}{2}(n^2-1)}\right).$$

This is the same probability for all instances I in the support of $D_1^{zkc}[R,n]$. So, we need only bound the probability

$$Pr_{I \sim \mathsf{Generate}(n,1)}[I \notin \text{ the support of } D_1^{zkc}[R,n]].$$

By Lemma 33 the initial process of choosing edges the probability of producing a clique is $\leq n^k/R$. We then change one edge's weight, this introduces a clique. It introduces an expected number of additional cliques $\leq n^{k-2}/R$ (this is the number of cliques it participates in). Thus, we can bound the probability of more than one clique by $\leq n^k/R + n^{k-2}/R$. □

Theorem 36. *Assuming the weak Zero-k-Clique hypothesis (ACIH) over range R, Zero-k-Clique is $(O(n^2), 2n^k/R)$-Plantable.*
Assuming the strong Zero-k-Clique hypothesis (ACSH) over range R, Zero-k-Clique is also $(O(n^2), 2n^k/R)$-Plantable.

Proof. This proof simply combines the two previous lemmas: Lemma 33 and Lemma 34.

$\mathsf{Generate}(n,0)$ has total variation distance n^k/R from $D_0^{zkc}[R,n]$ by Lemma 33, and $\mathsf{Generate}(n,1)$ has total variation distance $n^k/R + n^{k-2}/R < 2n^k/R$ from $D_1^{zkc}[R,n]$ by Lemma 34. So, in both cases the error is bounded above by $2n^k/R$.

Finally note that $\mathsf{Generate}(n,1)$ also can output the planted solution, the clique it chose to set to 0, and so can output a witness. □

4.9.3 Zero-k-Clique is Plantable, Average Case List-Hard and, Splittable from the Strong Zero-k-Clique Hypothesis

Here we will focus on the Strong Zero-k-Clique assumption, see Definition 44. Recall that this is the search version of the problem: given a graph with weights on its edges drawn uniformly from the k-partite graphs with exactly one zero k-clique, it is difficult to find the clique in time less than $\tilde{O}(n^k)$.

We already proved that Zero-k-Clique was Plantable in Theorem 35. So, now we will focus on the other two properties we want: list-hardness and splittability. These will give us the properties we need for our key exchange.

Zero-k-Clique is Average Case List-Hard

We present the proof that Zero-k-Clique is average case list-hard.

The intuition of the proof is as follows. There is an efficient worst case self-reduction for the Zero-k-Clique problem. This self-reduction results in $\ell'(n)^k$ sub-problems of size $n/\ell'(n)$. One can choose $\ell'(n)$ of these instances such that they are generated from non-overlapping parts of the original instance. They will then look uniformly randomly generated.

Now we will have generated many, $(\ell'(n))^k$, of these list versions of the Zero-k-Clique problem, where only one of them has the unique solution. We show that the problem is Average Case List-Hard by demonstrating that we can make many independent calls to the algorithm despite correlations between the instances called. Specifically, we only care about the response on one of these instances, so as long as that instance is random then we can solve the original problem.

Theorem 37. *Given the strong Zero-k-Clique-R Hypothesis , Zero-k-Clique is Average Case List-Hard with list length $\ell(n)$ for any $\ell(n) = n^{\Omega(1)}$.*

Proof. Let $\ell = \ell(n)$ for the sake of notation. $I \sim D_1(\text{Zero-}k\text{-Clique}, \ell \cdot n)$ with k partitions, $P_1, \ldots, P_k$ of $\ell \cdot n$ nodes each and with edge weights generated uniformly at random from $\mathbb{Z}_R$.

Randomly partition each P_i into ℓ sets $P_i^1, \ldots, P_i^\ell$ where each set contains n nodes. Now, note that if we look for a solution in all ℓ^k instances formed by taking every possible choice of $P_1^{i_1}, P_2^{i_2}, \ldots, P_k^{i_k}$, this takes time $O((\ell \cdot n)^k)$, which is how long the original size ℓn problem takes to solve.

Sadly, not all ℓ^k instances are independent. We want to generate sets of independent instances. Note that if we choose ℓ of these sub-problems such that the nodes don't overlap, then the edges were chosen independently between each instance! Specifically consider all vectors of the form $\mathbf{x} = \langle x_2, \ldots, x_k \rangle \in \mathbb{Z}_\ell^{k-1}$. Then let

$$S_{\mathbf{x}} = \{P_1^i \cup P_2^{i+x_2} \cup \ldots \cup P_k^{i+x_k} | \forall i \in [1, \ell]\}$$

be the set of all independent partitions. Now, note that $\cup_{\mathbf{x} \in \mathbb{Z}_\ell^{k-1}} S_{\mathbf{x}}$ is the full set of all possible ℓ^k subproblems, and the total number of problems in all $S_{\mathbf{x}}$ is ℓ^k, so once again brute-forcing each $S_{\mathbf{x}}$ takes time $O((\ell \cdot n)^k)$. We depict this splitting in Figure 4-3.

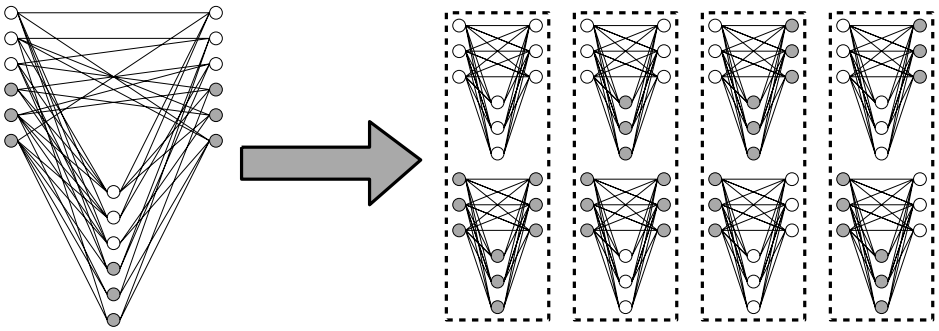

Figure 4-3: A depiction of splitting the subproblems for a case where $\ell = 2$ and $k = 3$.

Note that producing these each of these ℓ instances is efficient, it takes time $O(n^2)$, which is just the input size.

Next, we will show that the correct number of solutions are generated. If I has only one solution then exactly one I_j in exactly one $S_{\mathbf{x}}$ has a solution. This is because any zero-k-clique in I must involve exactly one node from each partition P_i. So, if

there is one zero-k-clique it will only appear in subproblems where the node from partition P_i is in P_i^j and P_j^i appears in that subproblem. There is exactly one subproblem generated with a specific choice of k sub-partitions. So, exactly one I_j in exactly one $S_{\mathbf{x}}$ has a solution.

Let S^* be the list $S_{\mathbf{x}}$ that contains a Zero-k-Clique. We have that the $S_{\mathbf{x}}$ which actually contains an instance with a solution is drawn from

$$\{I_1, \ldots, I_x\}_{I_i \sim D_1, \wedge \forall j \neq i, I_j \sim D_0}.$$

This distribution is exactly what we require for a list-problem. All that is left to show is if we have $\mathsf{PFT}_{\ell \cdot n^k}$ adversary $\mathcal{A}$ that can identify for which index i there is a zero k-clique in S^* (with probability at least 7/10), we can use $\mathcal{A}$ to find the clique.

Now, recall that we are trying to solve a search problem: we need to be able to turn an index pointing to partitions into a witness for the original problem. According to the strong Zero-k-Clique hypothesis, this search requires $O(n^k)$ time. However, as long as $\ell = n^{\Omega(1)}$, this is still faster in a fine-grained sense.

On an input I from D_1, algorithm $\mathcal{B}$ uses $\mathcal{A}$ as follows:

- Randomly partition each P_i from I into ℓ parts.
- For every $\mathbf{x} \in \mathbb{Z}_\ell^{k-1}$:
  - Generate the list $S_{\mathbf{x}}$.
  - Run $\mathcal{A}(S_{\mathbf{x}})$ to get output i.
  - Brute force search the size-n^2 Zero-k-Clique instance $S_{\mathbf{x}}[i] = (P_1^i, \ldots, P_k^{i+x_k})$ for a solution. If one exists, output it, otherwise, continue.

The first step only takes $O(\ell \cdot n)$ time since we are only divvying up ℓn nodes. The second step requires a bit more analysis. The loop runs at most ℓ^{k-1} times. Each time the loop runs, it only takes $O(\ell \cdot n)$ time to construct $S_{\mathbf{x}}$, while $\mathcal{A}$ takes $O((\ell \cdot n^k)^{(1-\epsilon)})$ (since it is $\mathsf{PFT}_{\ell \cdot n^k}$), and our brute force check takes $O(n^k)$ time. Putting this together, the algorithm takes a total time of

$$O(\ell \cdot n + \ell^{k-1}((\ell \cdot n^k)^{(1-\epsilon)} + n^k + \ell \cdot n)) = O(\ell^{k-\epsilon} n^{k(1-\epsilon)} + \ell^{k-1} n^k) + \ell^k \cdot n.$$

Both terms in this sum are strictly less than the hypothesized $\ell^k n^k$ time, and so $\mathcal{B}$ is $\mathsf{PFT}_{(\ell n)^k}$, contradicting the strong Zero-k-Clique hypothesis.

The reason we require $\ell(n) = n^{\Omega(1)}$ is because if it were less than polynomial in n, we would not get noticeable improvement through this method of splitting up the problem into several sub-problems — the brute force step would take as long as solving the original problem via brute force. □ □

Zero-k-Clique is Splittable

Next we show that zero-k-clique is splittable. We start by proving this for a convenient range and then show we can use a reduction to get more arbitrary ranges.

Splitting the problem over a convenient range. *Intuitively we will split the weights in half bit-wise, taking the first half of the bits of each edge weight, and then we take the second half of the bits of each edge weight to make another instance. If the $\binom{k}{2}$ weights on a k clique sum to zero then the first half of all the weights sum to zero, up to carries, and the second half of all the weights sum to zero, also up to carries. We simply guess the carries.*

Lemma 35. *Zero-k-Clique is generalized splittable with error $\leq 4(\binom{k}{2}+1)n^k/\sqrt{R}$ when $R = 4^x$ for some integer x.*

Proof. We are given an instance of Zero-k-Clique I with k partitions, $P_1, \ldots, P_k$ of n nodes and with edge weights generated uniformly at random from $[0, R-1]$, where $R = 2^{2x}$ for some positive integer x.

First we will define some helpful notation to describe our procedure.

- Let ZkC$[R]$ denote the Zero-k-Clique problem over range R.
- Let $w(P_i[a], P_j[b])$ be the weight of the edge in instance I between the a^{th} node in P_i and the b^{th} node in P_j.
- Let u be some number in the range $[0, R-1]$. Let $u^{\uparrow}$ be the high order $\lg(R)/2$ bits of the number u (this will be an integer because R is a power of 4). Let $u_{\downarrow}$ be the low order $\lg(R)/2$ bits of the number u.
  For the sake of notation, $w^{\uparrow}(P_i[a], P_j[b])$ denotes $[w(P_i[a], P_j[b])]^{\uparrow}$, and same for $w_{\downarrow}(P_i[a], P_j[b])$ denoting $[w(P_i[a], P_j[b])]_{\downarrow}$.

Here is the reduction to take one instance of ZkC$[R]$ and create a list of pairs of instances of ZkC$[\sqrt{R}]$.

1. Take the ZkC$[R]$ instance I and create two instances of ZkC$[\sqrt{R}]$, I_{low} and I_{high} by the following:
   - For every edge $(P_i[a], P_j[b])$ in I, let the corresponding edge in I_{low} have weight $w_{\downarrow}(P_i[a], P_j[b])$ and the edge in I_{high} have weight $w^{\uparrow}(P_i[a], P_j[b])$.
2. For every $c \in [0, \binom{k}{2}]$ (we need only check $\binom{k}{2}$ possible carries):
   (a) Let I_1^c be a copy of I_{low}, but randomly permute all nodes.
   (b) Let I_2^c be a copy of I_{high}, but choose a random pair of a partitions P_i and P_j: for all edges $e_2 \in I_2^c$ between P_i and P_j, a copy of edge $e \in I_{high}$, let $w(e_2) = w(e) + c \mod \sqrt{R}$.
3. Output the list $[(I_1^{(0)}, I_2^{(0)}), \ldots, (I_1^{(\binom{k}{2})}, I_2^{(\binom{k}{2})})]$

For a visual aid, see figure 4-4 for a depiction of the splittable triangles.

We will now show that we get the desired distributions in our list of instances depending on whether $I \sim D_0(\text{ZkC}[R], n)$ or $I \sim D_1(\text{ZkC}[R], n)$.

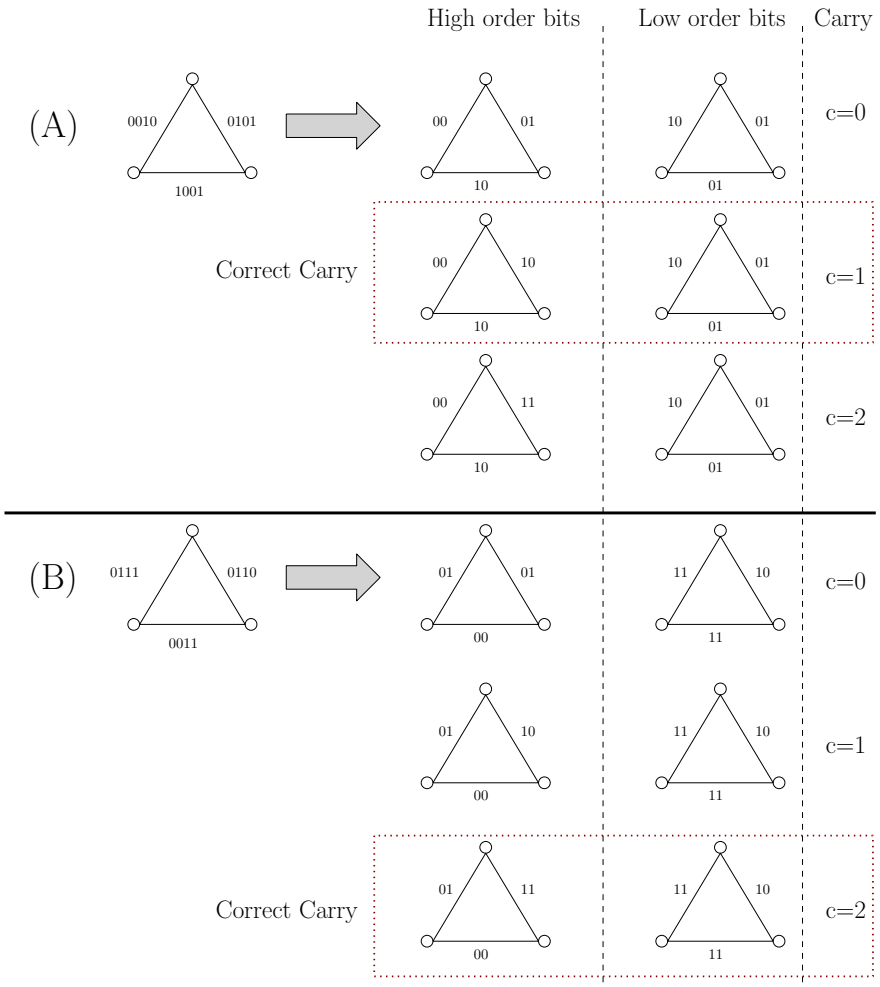

Figure 4-4: An example of splitting the edges of triangles whose edges sum to 16.

- $I \sim D_0(\text{ZkC}[R], n)$. We need to show that every pair $(I_1^{(c)}, I_2^{(c)})$ is sampled from a distribution total variation distance $\leq 2n^k/\sqrt{R}$ from $D_0(\text{ZkC}[\sqrt{R}], n)^2$. Note that every pair is correlated very heavily with every other pair with respect to edge weights. But, within each pair, they are close to $D_0(\text{ZkC}[\sqrt{R}], n)^2$.

  From lemma 33, this is TVD at most $\frac{n^k}{R}$ from just choosing edge-weights uniformly at random. So, consider $I' \sim \mathsf{Generate}(n, 0)$, and do the same operations as for I in the reduction: every bit in every edge weight will be chosen uniformly at random, meaning that the edge-weights in I'_{low} and I'_{high} will also be uniform over $\sqrt{R}$. Permuting the nodes in I'_{low} does not change this distribution, and neither does adding (any) c to a subset of edges in I'_{high}. Therefore, using lemma 33, *both* $I_1'^{(c)}$ and $I_2'^{(c)}$ are TVD at most $\frac{n^k}{\sqrt{R}}$ from $D_0(\text{ZkC}[\sqrt{R}], n)$. Since TVD is a metric, this implies that $I_1^{(c)}$ is TVD at most $n^k/\sqrt{R}$ from the distribution of $I_1'^{(c)}$, and thus at most $n^k/\sqrt{R} + n^k/R$ from $D_0(\text{ZkC}[\sqrt{R}], n)$ — the same is true for $I_2^{(c)}$, even when conditioned on $I_1^{(c)}$. Therefore, the pair, for every c, is TVD at most $2(n^k/\sqrt{R} + n^k/R) \leq \frac{4n^k}{\sqrt{R}}$.

- $I \sim D_1(\text{ZkC}[R], n)$. We want to show that we get a list in which exactly one of the pairs of instances is distributed close to $D_1(\text{ZkC}[\sqrt{R}], n)^2$.

  We will take a similar approach here, considering the planted distribution of I instead of the true one. Let $I' \sim \mathsf{Generate}(n, 1)$, so by lemma 34, I' is TVD at most $2n^k/R$ from D_1. We will first show that I'_{low} is also drawn from a planted distribution over the range $\sqrt{R}$. Let e' be the edge's weight that was changed to plant a zero clique. Now, for every edge except e'_{low}, the edges of I'_{low} are distributed uniformly. e' is a randomly chosen edge corresponding to a randomly chosen clique in I', and therefore e'_{low} is also a randomly chosen edge corresponding to a randomly chosen clique in I'_{low}. The act of making that clique sum to 0 mod R also requires that the low-order bits sum to 0 mod $\sqrt{R}$ — otherwise the high-order bits cannot cancel out anything left over. Therefore, by setting $w(e')$ to the value making the clique sum to 0, we are exactly planting a clique in I'_{low}. This distribution has TVD $\leq \frac{2n^k}{\sqrt{R}}$ from $D_1(\text{ZkC}[\sqrt{R}], n)$. Because $I_1'^{(c)}$ is just a permutation on the nodes of I'_{low} for every c, $I_1'^{(c)}$ will have TVD at most $\frac{2n^k}{\sqrt{R}}$ from D_1 as well.

  Now, we need that at least one of the pairs in this list to be close to $D_1(\text{ZkC}[\sqrt{R}], n) \times D_1(\text{ZkC}[\sqrt{R}], n)$. It will turn out that there exists a c such that $I_2'^{(c)}$ will also be close to D_1 (whereas $I_1'^{(c)}$ is distributed close to D_1 for every c). Let c^* be the correct carry — that is for the clique planted in I', $\sum_{e \in clique} w_\downarrow(e) = \sqrt{R}c^* \mod R$. Now, without loss of generality, we can assume that in the plant of I', the edge e^* chosen to complete the zero-k clique was between partitions P_i and P_j. So, considering every other edge in $I_2'^{(c^*)}$, it is distributed uniformly at random (adding c^* will not change that distribution). Now, for that special clique C^* that was planted in I', we have that

$$\begin{aligned}\sum_{e \in C^*} w(e) &= \sqrt{R} \cdot \sum_{e \in C^*} w^\uparrow(e) + \sum_{e \in C^*} w_\downarrow(e) \\ &= \sqrt{R}(\sum_{e \in C^*} w^\uparrow(e) + c^*) \\ &= \sqrt{R}(w^\uparrow(e^*) + c^* + \sum_{e \in C^*, e \neq e^*} w^\uparrow(e)) = 0 \mod R\end{aligned}$$

  Since the quantity $\sqrt{R}(w^\uparrow(e^*) + c^* + \sum_{e \in C^*, e \neq e^*} w^\uparrow(e))$ is 0 mod R, then $w^\uparrow(e^*) + c^* + \sum_{e \in C^*, e \neq e^*} w^\uparrow(e) = 0 \mod \sqrt{R}$.

  This means that $I_2'^{(c^*)}$ is drawn from $\mathsf{Generate}(n, 1)$ over the range $\sqrt{R}$. Since TVD is a metric, we have that for $I \sim D_1(\text{ZkC}[\sqrt{R}], n)$ (TVD at most $\frac{n^k}{R}$ from $\mathsf{Generate}(n, 1)$), there exists a c^* such that $I_2^{(c^*)}$ is TVD at most $\frac{2n^k}{\sqrt{R}}$ from D_1 — even when dependent on $I_1^{(c^*)}$. Therefore, the TVD of $(I_1^{(c^*)}, I_2^{(c^*)}) = \mathsf{Split}(I)$ to D_1^2 is at most $\frac{4n^k}{\sqrt{R}}$.

Therefore, when $I \sim D_0(\text{ZkC}[R], n)$, we get a list of pairs of instances TVD $\leq 4n^k/\sqrt{R}$ from $D_0(\text{ZkC}[R], n)^2$; the probability that any of these pairs here err is $\leq (\binom{k}{2}+1) \cdot \frac{4n^k}{\sqrt{R}}$

by a union bound. Similarly, when $I \sim D_1(\text{ZkC}[R], n)$, we get there exists a pair in this list of the form $D_1(\text{ZkC}[R], n)^2$; the probability of erring here is $\leq \frac{4n^k}{\sqrt{R}}$.

Therefore, the total error here is $\leq (\binom{k}{2} + 1) \cdot \frac{4n^k}{\sqrt{R}}$. □ □

Zero-k-Clique is Splittable Over Any Large Enough Range. *Our techniques also generalize to any large enough range (even ones not of the form 4^x). For example, if you believe that the problem is hard only over a prime range, we can prove that as well. As stated, our error is $\binom{k}{2}4^{\binom{k}{2}}3n^k/\sqrt{R} = O(n^k/\sqrt{R})$. For this to be meaningful, $R = \Omega(n^{2k})$, and in our constructions, R is $\Omega(n^{6k})$. We will show in the next section why the zero k-clique problem is still hard over these larger ranges.*

Theorem 38. *Zero-k-clique is generalized splittable over any range R, with error $\leq \binom{k}{2}4^{\binom{k}{2}}3n^k/\sqrt{R}$.*

Proof. Given an instance I with range R we will produce $\leq \binom{k}{2}4^{\binom{k}{2}}$ instances, corresponding to guesses over what ranges the clique edge weights fall into.

Take the next smallest power $R' = \max\{2^{2x} | 2^{2x} < R \text{ and } x \in \mathbb{Z}\}$. Now let $c = \lceil R/R' \rceil$. We will now create c subsets of R each of size R'. $S_i = [R'i, R'(i+1)-1]$ for $i \in [0, c-2]$ and $S_{c-1} = [R - R', R - 1]$. Note that these subsets completely cover the range $[0, R-1]$ and are each of size $\leq R'$. Let $\Delta_i = R'i$ for $i \in [0, c-2]$ and $\Delta_{c-1} = R - R'$.

Let the partitions of I be $P_1, \ldots, P_k$. Let the set of edges between P_i and P_j be $E_{i,j}$. For all i, j pairs $i \neq j$ we will choose a number between $[0, c-1]$. Call these numbers $g_{i,j}$ and the full list of them $\mathbf{g}$. For all possible choices of $\mathbf{g} \in \mathbb{Z}_c^{\binom{k}{2}}$ and $d \in [0, \binom{k}{2} - 1]$ we will generate an instance $I_{\mathbf{g},d}$ over range R' as follows:

For edge set $E_{i,j}$ that isn't $E_{1,2}$, for every edge in that edge set $e \in E_{i,j}$ if the weight of e, $w(e) \in S_{g_{i,j}}$ then set $w_{\mathbf{g},d}(e) = w(e) \mod R'$, if $w(e) \notin S_{g_{i,j}}$ then set $w_{\mathbf{g},d}$ to be a weight chosen uniformly at random from $[0, R'-1]$. Now note that these values are completely uniform over the range from $[0, R'-1]$.

For $E_{1,2}$, for every edge in that edge set $e \in E_{1,2}$ if the weight of e, $w(e) \in S_{g_{1,2}}$ then set $w_{\mathbf{g},d}(e) = w(e) + dR \mod R'$, if $w(e) \notin S_{g_{1,2}}$ then set $w_{\mathbf{g}}$ to be a weight chosen uniformly at random from $[0, R'-1]$. Now note that these values are also completely uniform over the range from $[0, R'-1]$.

If no clique existed in the original instance then the chance that one is produced here is bounded by $n^k/R' \leq n^k 4/R'$ by Lemma 33. Because we make so many queries this chance that any of them induce a clique is $\leq \binom{k}{2}4^{\binom{k}{2}}n^k 4/R'$.

If the original instance was drawn from $D_1^{zkc}[R, n]$ then by Lemma 34 this is only $\leq n^k/R + n^{k-2}/R$ total variation distance away from the instance generated by choosing each edge at random and then planting a clique. Then the procedure produces uniformly looking edges except for the planted edge. In the generated instance where the original zero clique edge weights are in $\mathbf{g}$ and the zero k-clique sums to dR then the instance $I_{\mathbf{g},d}$ will have that planted edge set to the value such that zero k-clique from the original is a planted instance. So, that produced instance

is drawn from a distribution with total variation distance $\leq n^k/R' + n^{k-2}/R'$ from $D_1^{zkc}[R', n]$.

Then we use the splitting procedure from Lemma 35 to generate two instances from each of our generated instances. The probability of a no instance becoming a yes instance is $\leq \binom{k}{2}4^{\binom{k}{2}}3n^k/\sqrt{R}$, if there is a yes instance then it will generate a yes instance and have total variation distance at most $\leq \binom{k}{2}4^{\binom{k}{2}}3n^k/\sqrt{R}$ from $D_1^{zkc}[\sqrt{R'}, n]$[2]. □ □

4.9.4 Larger Ranges for Zero-k-Clique are as Hard as Smaller Ranges.

We note that our constructions work for a large range: $R = \Omega(n^{6k})$. This theorem states that finding zero k-cliques over smaller ranges is as hard as finding them over larger ones. So, if you believe that Zero-k-Clique is hard over a range where an uniformly sampled instance is expected to have one solution (e.g. a small range like $O(n^k)$), we can show that this implies larger ranges, which are used extensively in our constructions, are also hard.

Theorem 39. *Strong Zero-k-Clique-R Hypothesis for range $R = n^{ck}$ is implied by the Random Edge Zero-k-Clique Hypothesis if $c > 1$ is a constant.*

Proof. Create $n^{(1-1/c)}$ random partitions of the nodes where each partition is of size $n^{1/c}$. Then generate $n^{(1-1/c)k}$ graphs by choosing every possible choice of k partitions.

This results in $n^{(1-1/c)k}$ problems of size $n^{1/c}$ with range n^k.

If an algorithm A violated Strong Zero-k-Clique-R Hypothesis for range n^{ck} then it must have some running time of the form $O(n^{k-\delta})$ for $\delta > 0$. We could run A on all $n^{(1-1/c)k}$ problems, resulting in a running time of $n^{k/c-\delta/c}n^{(1-1/c)}k = O(n^{k-\delta/c})$ for the Random Edge problem. If we find a valid zero-k-clique then we return 1. If we don't we return 0 with probability 1/2 and 1 with probability 1/2.

Let p_1 be the probability that any one of the $n^{k/c}$ has exactly one zero clique, conditioned on $val(I) = 1$ (that there is at least one solution).

If Strong Zero-k-Clique-R Hypothesis is violated then we return the correct answer with the probability at least $p_1\frac{1}{100} + (1 - p_1/100)/2 = 1/2 + p_1\frac{1}{200}$. So, we now want to lower bound the value of p_1.

The probability that there are more than 2 cliques in a subproblem of size $n^{1/c}$, conditioned on there being at least one clique is at most $n^{-k(1-1/c)}$. Because to generate the distribution of problems of size $n^{1/c}$ with at least one clique one can plant a clique (randomly choose k nodes and randomly choose $\binom{k}{2} - 1$ edge weights, then choose the final edge weight such that this is a zero k-clique). The expected number of cliques other than the planted clique is $\frac{n^{1/c}-1}{n^k}$ and the number of cliques other than the planted clique is a non-negative integer.

So conditioned $val(I) = 1$ we have that $p_1 \geq 1 - n^{-k(1-1/c)}$. So the probability of success, conditioned on $val(I) = 1$ is at least $p_1/100 \geq 1/200$.

□

Bibliography

[1] *Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In* International Colloquium on Automata, Languages, and Programming*, pages 39–51. Springer, 2014.*

[2] *Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In* 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014.*

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

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

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

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

[7] *Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In* Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 20–29, 1996.*

[8] *Noga Alon, Raphael Yuster, and Uri Zwick. Color coding. In* Encyclopedia of Algorithms*, pages 335–338. Springer, 2016.*

[9] *Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In* Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 171–180, New York, NY, USA, 2010. ACM.*

[10] *Benny Applebaum, Naama Haramaty, Yuval Ishai, Eyal Kushilevitz, and Vinod Vaikuntanathan. Low-complexity cryptographic hash functions. In* 8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 7:1–7:31, 2017.*

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

[12] *László Babai, Anna Gál, Peter G. Kimmel, and Satyanarayana V. Lokam. Communication complexity of simultaneous messages.* SIAM J. Comput.*, 33(1):137–166, 2003.*

[13] *Arturs Backurs and Christos Tzamos. Improving viterbi is hard: Better runtimes imply faster clique algorithms.* CoRR*, abs/1607.04229, 2016.*

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

[15] *Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-case fine-grained hardness. In* Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 483–496, 2017.*

[16] *Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of work from worst-case assumptions. In* Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, pages 789–819, 2018.*

[17] *Boaz Barak and Mohammad Mahmoody-Ghidary. Merkle puzzles are optimal - an* $O(n^2)$*-query attack on any key exchange from a random oracle. In* Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 374–390, 2009.*

[18] *Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3sum.* Algorithmica*, 50(4):584–596, 2008.*

[19] *Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In* Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, pages 566–582, 2001.*

[20] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In 37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 514–523, 1996.*

[21] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In 38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 394–403, 1997.*

[22] Mihir Bellare, Roch Guérin, and Phillip Rogaway. XOR macs: New methods for message authentication using finite pseudorandom functions. In Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, pages 15–28, 1995.*

[23] Mihir Bellare, Russell Impagliazzo, and Moni Naor. Does parallel repetition lower the error in computationally sound protocols? In Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, pages 374–, Washington, DC, USA, 1997. IEEE Computer Society.*

[24] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of cipher block chaining. In Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, pages 341–358, 1994.*

[25] Mihir Bellare and Thomas Ristenpart. Simulation without the artificial abort: Simplified proof and improved concrete security for waters' IBE scheme. In Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 407–424, 2009.*

[26] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In 25th ACM STOC*, pages 52–61. ACM Press, May 1993.*

[27] Eli Biham, Yaron J. Goren, and Yuval Ishai. Basing weak public-key cryptography on strong one-way functions. In Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008.*, pages 55–72, 2008.*

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

[29] Elette Boyle, Rio LaVigne, and Vinod Vaikuntanathan. Adversarially robust property-preserving hash functions. In 10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 16:1–16:20, 2019.*

[30] *Karl Bringmann, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless APSP can). In* Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1190–1206, 2018.*

[31] *Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In* 42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.*

[32] *Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors,* TCC 2015, Part II*, volume 9015 of* LNCS*, pages 468–497. Springer, Heidelberg, March 2015.*

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

[34] *Michael Capalbo, Omer Reingold, Salil Vadhan, and Avi Wigderson. Randomness conductors and constant-degree lossless expanders. In* Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 659–668, New York, NY, USA, 2002. ACM.*

[35] *Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In* Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 261–270, 2016.*

[36] *Larry Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In* Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 106–112, 1977.*

[37] *Timothy M. Chan. More logarithmic-factor speedups for 3sum, (median, +)-convolution, and some geometric 3sum-hard problems. In* Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 881–897, 2018.*

[38] *Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for UC secure computation using tamper-proof hardware. In* Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 545–562, 2008.*

[39] *Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams.* Theor. Comput. Sci.*, 312(1):3–15, 2004.*

[40] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM*, 24(2):84–88, 1981.*

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

[42] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of dual EC in TLS implementations. In Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 319–335, 2014.*

[43] Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic decomposition by basis pursuit. SIAM Rev.*, 43(1):129–159, 2001.*

[44] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, ICS 2010*, pages 310–331. Tsinghua University Press, January 2010.*

[45] Seung Geol Choi, Jonathan Katz, Dominique Schröder, Arkady Yerukhimovich, and Hong-Sheng Zhou. (Efficient) universally composable oblivious transfer using a minimal number of stateless tokens. In Yehuda Lindell, editor, TCC 2014*, volume 8349 of* LNCS*, pages 638–662. Springer, Heidelberg, February 2014.*

[46] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. J. ACM*, 45(6):965–981, 1998.*

[47] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. In Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, pages 240–269, 2016.*

[48] Colin Cooper, Alan M. Frieze, Kurt Mehlhorn, and Volker Priebe. Average-case complexity of shortest-paths problems in the vertex-potential model. Random Struct. Algorithms*, 16(1):33–46, 2000.*

[49] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. J. Algorithms*, 55(1):58–75, 2005.*

[50] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 378–394, 2005.*

[51] Akshay Degwekar, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Fine-grained cryptography. In Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, pages 533–562, 2016.*

[52] *W. Diffie and M. Hellman. New directions in cryptography.* IEEE Trans. Inf. Theor., *22(6):644–654, September 2006.*

[53] *Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data.* SIAM J. Comput., *38(1):97–139, March 2008.*

[54] *Bella Dubrov and Yuval Ishai. On the randomness complexity of efficient sampling. In Jon M. Kleinberg, editor,* 38th ACM STOC, *pages 711–720. ACM Press, May 2006.*

[55] *I. Dumer, A. A. Kovalev, and L. P. Pryadko. Distance verification for ldpc codes. In* 2016 IEEE International Symposium on Information Theory (ISIT), *pages 2529–2533, July 2016.*

[56] *Cynthia Dwork, Moni Naor, and Omer Reingold. Immunizing encryption schemes from decryption errors. In Christian Cachin and Jan L. Camenisch, editors,* Advances in Cryptology - EUROCRYPT 2004, *pages 342–360, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.*

[57] *H Fleischner. X. 1 algorithms for eulerian trails.* Eulerian Graphs and Related Topics: Part 1 (Annals of Discrete Mathematics), *2(50):1–13, 1991.*

[58] *M Fleury. Deux problèmes de géométrie de situation.* Journal de mathématiques élémentaires, *2:257–261, 1883.*

[59] *Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry.* Comput. Geom., *45(4):140–152, 2012.*

[60] *François Le Gall. Powers of tensors and fast matrix multiplication. In* International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014, *pages 296–303, 2014.*

[61] *Robert Gallagher. Low-density parity-check codes, 1963.*

[62] *Leonid Geller and David Burshtein. Bounds on the belief propagation threshold of non-binary LDPC codes.* IEEE Trans. Information Theory, *62(5):2639–2657, 2016.*

[63] *O. Goldreich, R. Impagliazzo, L. Levin, R. Venkatesan, and D. Zuckerman. Security preserving amplification of hardness. In* Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science, *pages 318–326 vol.1, Oct 1990.*

[64] *O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In* Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing, *STOC '89, pages 25–32, New York, NY, USA, 1989. ACM.*

[65] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. J. ACM*, 33(4):792–807, 1986.*

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

[67] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, pages 308–326, 2010.*

[68] Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 121–130, 2013.*

[69] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. Transactions of the American Mathematical Society*, 117:285–306, 1965.*

[70] Elad Hazan and Robert Krauthgamer. How hard is it to approximate the best nash equilibrium? SIAM J. Comput.*, 40(1):79–91, 2011.*

[71] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 205–220, 2012.*

[72] F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape turing machines. J. ACM*, 13(4):533–546, October 1966.*

[73] Markus Hinkelmann and Andreas Jakoby. Communications in unknown networks: Preserving the secret of topology. Theor. Comput. Sci.*, 384(2-3):184–200, 2007.*

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

[75] Martin Hirt, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Network-hiding communication and applications to multi-party protocols. In Matthew Robshaw and Jonathan Katz, editors, CRYPTO 2016, Part II*, volume 9815 of* LNCS*, pages 335–365. Springer, Heidelberg, August 2016.*

[76] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In IN

PROC. OF THE 5TH CENTRAL EUROPEAN CONFERENCE ON CRYPTOLOGY MORAVIACRYPT 2005*, 2005.*

[77] Russell Impagliazzo. A personal view of average-case complexity. In Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*, pages 134–147, 1995.*

[78] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. Journal of Cryptology*, 9, 02 2002.*

[79] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In 41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 189–197, 2000.*

[80] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 604–613, 1998.*

[81] T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. Theory of Computing*, 4(1):129–135, 2008.*

[82] Mark Jerrum. Large cliques elude the metropolis process. Random Struct. Algorithms*, 3(4):347–360, 1992.*

[83] Ari Juels and Marcus Peinado. Hiding cliques for cryptographic security. Designs, Codes and Cryptography*, 20(3):269–280, Jul 2000.*

[84] Daniel M. Kane and R. Ryan Williams. The orthogonal vectors conjecture for branching programs and formulas. CoRR*, abs/1709.05294, 2017.*

[85] Harini Kannan, Alexey Kurakin, and Ian J. Goodfellow. Adversarial logit pairing. CoRR*, abs/1803.06373, 2018.*

[86] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, pages 477–498, 2013.*

[87] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, Washington, DC, USA, October 27-30, 2003*, pages 155–164, 2003.*

[88] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1272–1287, 2016.*

[89] *Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. In* Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 596–605, New York, NY, USA, 1995. ACM.*

[90] *Ludek Kucera. Expected complexity of graph partitioning problems.* Discrete Applied Mathematics*, 57(2-3):193–212, 1995.*

[91] *Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In* Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 614–623, New York, NY, USA, 1998. ACM.*

[92] *Rio LaVigne. Topology hiding computation on all graphs. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2017.*

[93] *Rio LaVigne, Andrea Lincoln, and Virginia Vassilevska Williams. Public-key cryptography in the fine-grained setting.* IACR Cryptology ePrint Archive*, 2019:625, 2019.*

[94] *Rio LaVigne, Chen-Da Liu Zhang, Ueli Maurer, Tal Moran, Marta Mularczyk, and Daniel Tschudi. Topology-hiding computation beyond semi-honest adversaries. In* Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part II*, pages 3–35, 2018.*

[95] *Rio LaVigne, Chen-Da Liu Zhang, Ueli Maurer, Tal Moran, Marta Mularczyk, and Daniel Tschudi. Topology-hiding computation for networks with unknown delays. In* To appear in Public Key Cryptography PKC 2020, Edinburgh, Scotland, May 4-7, 2020*, 2020.*

[96] *Leonid A. Levin. On storage capacity of algorithms.* Soviet Mathematics, Doklady*, 14(5):1464–1466, 1973.*

[97] *Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In* Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252, 2018.*

[98] *Yehuda Lindell.* Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*. Springer Publishing Company, Incorporated, 1st edition, 2017.*

[99] *Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In Rosario Gennaro and Matthew J. B. Robshaw, editors,* CRYPTO 2015, Part II*, volume 9216 of* LNCS*, pages 319–338. Springer, Heidelberg, August 2015.*

[100] *S. Litsyn and V. Shevelev. On ensembles of low-density parity-check codes: asymptotic distance distributions.* IEEE Transactions on Information Theory*, 48(4):887–908, Apr 2002.*

[101] *Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In* Proceedings of the 7th International Conference on Theory of Cryptography*, TCC'10, pages 382–400, Berlin, Heidelberg, 2010. Springer-Verlag.*

[102] *Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks.* CoRR*, abs/1706.06083, 2017.*

[103] *S. A. M. Makki. A distributed algorithm for constructing an eulerian tour. In* 1997 IEEE International Performance, Computing and Communications Conference*, pages 94–100, Feb 1997.*

[104] *Ralph C. Merkle. Secure communications over insecure channels.* Commun. ACM*, 21(4):294–299, April 1978.*

[105] *Ilya Mironov, Moni Naor, and Gil Segev. Sketching in adversarial environments. In* Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 651–660, 2008.*

[106] *Jayadev Misra and David Gries. Finding repeated elements.* Sci. Comput. Program.*, 2(2):143–152, 1982.*

[107] *Tal Moran, Ilan Orlov, and Silas Richelson. Topology-hiding computation. In* Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, pages 159–181, 2015.*

[108] *Tal Moran, Ilan Orlov, and Silas Richelson. Topology-hiding computation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors,* TCC 2015, Part I*, volume 9014 of* LNCS*, pages 159–181. Springer, Heidelberg, March 2015.*

[109] *J. Ian Munro and Mike Paterson. Selection and sorting with limited storage.* Theor. Comput. Sci.*, 12:315–323, 1980.*

[110] *Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In* Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 565–584, 2015.*

[111] *Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In* Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 33–43, 1989.*

[112] *Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In* Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610, 2010.*

[113] *Yuval Peres, Dmitry Sotnikov, Benny Sudakov, and Uri Zwick. All-pairs shortest paths in* $O(n^2)$ *time with high probability.* J. ACM*, 60(4):26:1–26:25, 2013.*

[114] *Seth Pettie. Higher lower bounds from the 3sum conjecture. Fine-Grained Complexity and Algorithm Design Workshop at the Simons Institute, 2015.*

[115] *Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples.* CoRR*, abs/1801.09344, 2018.*

[116] *Sivaramakrishnan Natarajan Ramamoorthy and Makrand Sinha. On the communication complexity of greater-than. In* 53rd Annual Allerton Conference on Communication, Control, and Computing, Allerton 2015, Allerton Park & Retreat Center, Monticello, IL, USA, September 29 - October 2, 2015*, pages 442–444, 2015.*

[117] *Alexander A. Razborov and Steven Rudich. Natural proofs. In* Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 204–213, 1994.*

[118] *Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In* Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. ACM.*

[119] *Oded Regev. Lattice-based cryptography (invited talk). In Cynthia Dwork, editor,* CRYPTO 2006*, volume 4117 of* LNCS*, pages 131–141. Springer, Heidelberg, August 2006.*

[120] *Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions.* ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.*

[121] *R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems.* Commun. ACM*, 21(2):120–126, February 1978.*

[122] *Alexander Russell and Hong Wang. How to fool an unbounded adversary with a short key. In* Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '02, pages 133–148, London, UK, UK, 2002. Springer-Verlag.*

[123] *Adi Shamir. How to share a secret.* Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.*

[124] *P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In* Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, pages 124–134, Washington, DC, USA, 1994. IEEE Computer Society.*

[125] *Aman Sinha, Hongseok Namkoong, and John C. Duchi. Certifiable distributional robustness with principled adversarial training.* CoRR*, abs/1710.10571, 2017.*

[126] *Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In* 1997 IEEE Symposium on Security and Privacy, May 4-7, 1997, Oakland, CA, USA*, pages 44–54, 1997.*

[127] *G. S. Tseitin. On the complexity of derivation in propositional calculus. Presented in the Leningrad Seminar on Mathematical Logic, 1956.*

[128] *Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In* Proceedings of the International Congress of Mathematicians*, page to appear, 2018.*

[129] *Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In* Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898, 2012.*

[130] *Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In* 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654, 2010.*

[131] *Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs.* SIAM J. Comput.*, 42(3):831–854, 2013.*

[132] *David Woodruff. Optimal space lower bounds for all frequency moments. In* Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 167–175, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.*

[133] *David P. Woodruff.* Efficient and private distance approximation in the communication and streaming models*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2007.*

[134] *Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In* Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 209–213, 1979.*