Encrypted Distributed Hash Tables

Archita Agarwal* Brown University

Seny Kamara† Brown University

Abstract

Distributed hash tables (DHT) are a fundamental building block in the design of distributed systems with applications ranging from content distribution networks to off-chain storage networks for blockchains and smart contracts. When DHTs are used to store sensitive information, system designers use end-to-end encryption in order to guarantee the confidentiality of their data. A prominent example is Ethereum's off-chain network Swarm.

In this work, we initiate the study of end-to-end encryption in DHTs and the many systems they support. We introduce the notion of an *encrypted DHT* and provide simulation-based security definitions that capture the security properties one would desire from such a system. Using our definitions, we then analyze the security of a standard approach to storing encrypted data in DHTs. Interestingly, we show that this "standard scheme" leaks information *probabilistically*, where the probability is a function of how well the underlying DHT load balances its data. We also show that, in order to be securely used with the standard scheme, a DHT needs to satisfy a form of equivocation with respect to its overlay. To show that these properties are indeed achievable in practice, we study the balancing properties of the Chord DHT—arguably the most influential DHT—and show that it is equivocable with respect to its overlay in the random oracle model. Finally, we consider the problem of encrypted DHTs in the context of transient networks, where nodes are allowed to leave and join.

*archita_agarwal@brown.edu

†seny@brown.edu

Contents

1 Introduction

In the early 2000's, the field of distributed systems was revolutionized in large part by the performance and scalability requirements of large Internet companies like Akamai, Amazon, Google and Facebook. The operational requirements of these companies—which include running services at Internet scale using commodity hardware in data centers distributed across the world—motivated the design of highly influential systems like Chord [27], Dynamo [11] and BigTable [7]. These advances in distributed systems are what enable companies like Amazon to handle over a billion purchases a year and Facebook to support two billion users worldwide.

Distributed hash tables. The most fundamental building block in the design of highly scalable and reliable systems are *distributed hash tables* (DHT). DHTs are decentralized and distributed systems that store data items associated to a label. Roughly speaking, a DHT is a distributed dictionary data structure that stores label/value pairs (ℓ, v) and that supports get and put operations. The former takes as input a label ℓ and returns the associated value v. The latter takes as input a pair (ℓ, v) and stores it. DHTs are distributed in the sense that the pairs are stored by a set of n nodes $N_1, \ldots, N_n$. To communicate and route messages to and from nodes, DHTs rely on a (usually) randomly generated *overlay network* which, intuitively, maps node names to addresses and of a distributed routing protocol that routes messages between addresses. DHTs provide many useful properties but the most important are load balancing and fast data retrieval and storage even in highly-transient networks (i.e., where storage nodes join and leave at high rates).

Classic applications of DHTs. It is hard to overstate the impact that DHTs have had on system design and listing all their possible applications is not feasible so we will recall just a few. One of the first applications of DHTs was to the design of content distribution networks (CDNs). In 1997, Karger et al. introduced the notion of consistent hashing [17] which was adopted as a core component of Akamai's CDN. Since then, many academic and industry CDNs have used DHTs for fast content delivery [13, 26]. DHTs are also used by many P2P systems like BitTorrent [1] and its many trackerless clients including Vuze, rTorrent, Ktorrent and μTorrent. Many distributed file systems are built on top of DHTs, including CFS [10], Ivy [21], Pond [23], PAST [12].DHTs are also the main component of distributed key-value stores like Amazon's Dynamo [11] which underlies the Amazon cart, LinkedIn's Voldemort [28] and Riak [29]. Finally, many wide column NoSQL databases like Facebook's Cassandra [19], Google's BigTable [7] and Amazon's DynamoDB [25] make use of DHTs.

Off-chain storage. Currently, the field of distributed systems is going through another revolution brought about by the introduction of blockchains [22]. Roughly speaking, blockchains are distributed and decentralized storage networks with integrity and probabilistic eventual consistency. Blockchains have many interesting properties and have fueled an unprecedented amount of interest in distributed systems and cryptography. For all their appeal, blockchains have several shortcomings; the most important of which are limited storage capacity and lack of confidentiality. To address this, a lot of effort in recent years has turned to the design of distributed and/or decentralized *off-chain* storage networks whose primary purpose is to store large amounts of data while supporting fast retrieval and storage in highly transient networks. In fact, many influential blockchain projects, including Ethereum [31, 2], Enigma [32], Storj [24] and Filecoin [18] rely on off-chain storage: Ethereum, Enigma and Storj on their own custom networks and Filecoin on IPFS [3]. Due to the storage and scalability requirements of these blockchains, these off-chain storage networks often use DHTs as a core building block.

DHTs and end-to-end encryption. As discussed, DHTs are a fundamental building block in distributed systems with applications ranging from CDNs to blockchains. DHTs were originally designed for applications that mostly dealt with public data: for example, web caching or P2P file sharing. The more recent applications of DHTs, however, also need to handle *private* data. This is the case, for example, for off-chain storage networks, many of which aim to support decentralized apps for medical records, IoT data, tax information, customer records and insurance data, just to name a few. Indeed, most of these networks (e.g., Ethereum's Swarm, IPFS, Storj and Enigma) explicitly implement some form of end-to-end encryption.

The specific designs are varied but, as far as we know, none of them have been formally analyzed. This is not surprising, however, since the problem of end-to-end encryption in the context of DHTs has never been properly studied. In this work, we address this by formalizing the goals of encryption in DHTs. In particular, we introduce the notion of an *encrypted DHT* (EDHT) and propose formal syntax and security definitions for these objects. Due to the ubiquity of DHTs and the recent interest in using them to store sensitive data, we believe that a formal study of confidentiality in DHTs is a well-motivated problem of practical importance.

The standard scheme. The simplest approach to storing sensitive data on a DHT—and the one we will study in this work—is to store a label/value pair (ℓ, v) as $(F_{K_1}(\ell), \mathsf{Enc}_{K_2}(v))$ on a standard DHT. Here, F is a pseudo-random function and Enc is a symmetric-key encryption scheme. Throughout we will refer to this as the *standard scheme*. The underlying DHT will then assign this pair to a storage node in a load balanced manner, handle routing and will move pairs around the network if a node leaves or joins. This scheme is simple and easy to implement and is, roughly speaking, what most systems implement. Ethereum's Swarm, for example, stores pairs as $(H(\mathrm{ct}), \mathrm{ct})$, where $\mathrm{ct} \leftarrow \mathsf{Enc}_K(v)$ and H is a hash function. But is this secure? Answering this question is not simple as it is not even clear what we mean by security. But even if we were equipped with a meaningful notion of security, we will see that the answer is not straightforward. The reason is because, as we will see, the security of the standard scheme is tightly coupled with how the the underlying DHT is designed.

Information leakage in EDHTs. To illustrate this point, suppose a subset of nodes are corrupted and collude. During the operation of this DHT, what information can they learn about a client's data and/or queries? A-priori, it might seem that the only information they can learn is related to what they collectively hold (i.e., the union of the data they store). For example, they might learn that there are at least m pairs stored in the DHT, where m is the sum of the number of pairs held by each corrupted node. With respect to the client's queries they might learn, for any label handled by a corrupted node, when a query repeats. While this intuition might seem correct, it is not true. In fact, the corrupted nodes can infer additional information about data they do not hold. For example, they can infer a good approximation on the *total* number of pairs in the system even if they collectively hold a small fraction of it. Here, the problem is that DHTs are load balanced in the sense that, with high probability, each node will receive approximately the same number of pairs. Because of this, the corrupted nodes can guess that, with high probability, the total number of pairs in the system is about mn/t, where t is the number of corrupted nodes and n is the total number of nodes.

While this may seem benign, this is just one example to highlight the fact that finding and analyzing information leakage in distributed systems can be non-trivial. In fact, some of the very properties which we aim for in the context of distributed systems (e.g., load balancing) can have subtle effects on security.

1.1 Our Contributions

In this work, we aim to formalize the use of end-to-end encryption in DHTs and the many systems they support. As an increasing number of applications wish to store sensitive data on DHT-based systems, the use of end-to-end encryption in DHTs should be raised from a technique to a cryptographic primitive with formal syntax and security definitions. Equipped with these definitions, our goal will be to understand and study the security guarantees of the simple EDHT described above. As we will see, analyzing and proving the security of even this simple scheme is complex enough. We make several contributions.

Security of EDHTs. Our first contribution is a simulation-based definition of security for EDHTs. The definition is in the real/ideal-world paradigm commonly used to formalize the security of multi-party computation [5]. Formulating security in this way allows for definitions that are modular and intuitive. Furthermore, this seems to be a natural way to define security since DHTs are distributed objects. In our definition, we compare a real-world execution between n nodes, an honest client and an adversary, where the latter can corrupt a subset of the nodes. Roughly speaking, we say that an EDHT is secure if this experiment is indistinguishable from an ideal-world execution between the nodes, the honest client, an ideal adversary (i.e., a simulator) and an functionality that captures the ideal security properties of EDHTs. As discussed above, for any EDHT scheme, including the standard construction, there can be subtle ways in which some information about the dataset is leaked (e.g., its total size). To formally capture this, we parameterize our definition with (stateful) leakage functions that capture exactly what is or is not being revealed to the adversary. We note that our definitions handle static corruptions and are in the standalone setting.

EDHTs and structured encryption. The notion of an EDHT can be viewed and understood from the perspective of structured encryption (STE). STE schemes are encryption schemes that encrypt data structures in such a way that they can be privately queried. From this perspective, EDHTs are a form of distributed encrypted dictionaries and, in fact, one recovers the latter from the former when the network consists of only one node. We note that this connection is not just syntactical, but also holds with respect to the security definitions of both objects and to their leakage profiles. Indeed the standard scheme's leakage profile on a single-node network reduces to the leakage profile of common dictionary encryption schemes [8, 6]. This leakage, however, represents the "worst-case" leakage of the standard EDHT. This suggests that distributed STE schemes can leak less than non-distributed STE schemes which makes sense intuitively since, in the distributed setting, the adversary can only corrupt a subset of the nodes whereas in the non-distributed setting the adversary corrupts the only existing node and, therefore, all the nodes.

With this in mind, one can view our results as another approach to the recent efforts to suppress the leakage of STE schemes [16, 15]. That is, instead of (or in addition to) compiling STE schemes as in [16] or of transforming the underlying data structures as in [15], one could *distribute* the encrypted data structure.

Probabilistic leakage. Our security definition allows us to formally study any leakage produced by EDHT schemes. Interestingly, our analysis of the standard scheme will show that it achieves a very novel kind of leakage profile. Now, this leakage profile is itself quite interesting. First, it is *probabilistic* in the sense that it leaks only with some probability $p \leq 1$. As far as we know, this is the fist time such a leakage profile has been encountered. Here, the information it leaks (when it does leak) is the query equality pattern (see [16] for a discussion of various leakage patterns)

which reveals if and when a query was made in the past. This is not surprising as labels are passed as $F_K(\ell)$ to the underlying DHT, which are deterministic. This leakage profile is also interesting because the probability p with which it leaks is determined by properties of the underlying DHT and, in particular, to its load balancing properties. Specifically, the better the DHT load balances its data the smaller the probability that the EDHT will leak the query equality.

Worst-case vs. expected leakage. A-priori one might think that the adversary should only learn information related to pairs that are stored on corrupted nodes and that, since DHTs are load balanced, the total number of pairs visible to the adversary will be roughly mt/n. But there is a slight technical problem with this intuition: a DHT' s allocation of labels depends on its overlay and, for any set of corrupted nodes, there are many overlays that can induce an allocation where, say, a very large fraction of labels are mapped to corrupted nodes. The problem then is that, in the *worst-case*, the adversary could see *all* the (encrypted) pairs. We will show, however, that the intuition above is still correct because the worst-case is unlikely to occur. More precisely, we show that with probability at least $1-\delta$ over the choice of overlay, the standard scheme achieves a certain leakage profile $\mathcal{L}$ which is a function of δ (and other parameters). As far as we know, this is the first example of a leakage analysis that is not worst-case but that, instead, considers the expected leakage (with high probability) of a construction. We believe this new kind of leakage analysis is of independent interest and that the idea of expected leakage may be a fruitful direction in the design of low- or even zero-leakage schemes. In Section A, we show how to further reduce leakage using additional machinery.

Formalizing DHTs. To better understand EDHTs and their security properties, we aim for a modular treatment. In particular, we want to isolate the properties of the underlying DHTs that have an effect on security and decouple the components of the system that have to do with the DHT from the cryptographic primitives we use like encryption and PRFs. This is in line with how systems designers use encryption in DHTs; as far as we know, all DHT-based systems that support end-to-end encryption add encryption on top of an "unmodified" DHT. Our first step, therefore, is to formally define DHTs. This includes a formal syntax but, more interestingly, a useful abstraction of the core components of a DHT including, their network overlays, their allocations (i.e., how they map label/value pairs to nodes) and their routing components.

Properties of DHTs. As mentioned above, we found that the security of the standard EDHT scheme is tightly coupled with two main properties of DHTs. More precisely, we discovered that the former's leakage is affected by a property we call *balance* which, roughly speaking, means that with probability at least $1-\delta$ over the choice of overlays, the DHT allocates any label ℓ to any θ-sized set of nodes with probability at most ε (over the choice of allocation). Note that this definition essentially guarantees a (one-sided) form of load balancing.

Another interesting finding we made was that if the standard scheme is to satisfy our simulation-based definition, then the underlying DHT has to satisfy a form of equivocation. Intuitively, the DHT must be designed in such a way that, for any fixed overlay within a (large) class of overlays, it is possible to "program" the allocation so that it maps a given label to a given server. We found the appearance of equivocation in the context of DHTs quite surprising as it is usually a property that comes up in the context of cryptographic primitives.

Chord in the perpetual setting. Having isolated the properties we need from a DHT in order to prove the security of the standard scheme, it is natural to ask whether there are any known DHTs

that satisfy them. Interestingly, we not only found that such DHTs exist but that Chord [17]—which is arguably the most influential DHT—is both balanced and non-committing in the sense that it supports the kind of equivocation discussed above in the random oracle model. Without getting into details of how Chord works (we refer the reader to section 5 for a description), we mention here that Chord makes use of two hash functions: one to map names to addresses and a second to map labels to addresses. In section 5, we show that Chord is non-committing if the second hash function is modeled as a random oracle.

Transient EDHTs. All the analysis discussed above was for what we call the *perpetual* setting where nodes never leave the network. [1] Note that the perpetual setting is realistic and interesting in itself. It captures, for example, how DHTs are used by many large companies who run nodes in their own data centers, e.g., Amazon, Google, LinkedIn. Nevertheless, we also consider the *transient* setting where nodes are allowed to leave and join the network arbitrarily. We extend our syntax and security definitions to this setting and prove that the standard scheme—equipped with certain join and leave protocols—achieves another probabilistic leakage profile. Necessarily, this leakage profile is more complex than the one achieved in the perpetual setting. At a high level, it works as follows. For puts and gets the leakage is roughly the same as in the perpetual setting. For joins, it leaks the number of previous put operations for labels that were stored and routed exclusively by honest nodes. For leaves there are two cases. When an honest node leaves, the leakage is the same as a join and when a corrupted node leaves there is no leakage. Our leakage analysis in the transient setting relies on a new and stronger property of the underlying DHT we call *stability* which, roughly speaking, means that with probability at least $1 - \delta$ over the choice of overlay parameter ω, for all large enough overlays, the DHT allocates any label to any θ-sized set with probability at most ε.

Chord in the transient setting. Having analyzed the standard EDHT in the transient setting, we study its properties when it is instantiated with a transient variant of Chord. Our analysis of Chord's stability is non-trivial. At a very high level the main challenge is that, in the transient setting, Chord's overlay changes with every leave or join. To handle this, we introduce a series of (probabilistic) bounds to handle "dynamic" overlays that may be of independent interest.

Future applications. Because DHTs are a central building block in distributed systems, we expect EDHTs to become central building blocks in the design and analysis of *encrypted* distributed systems. We describe several examples in Section **??**.

Related work. Since we already discussed related work on DHTs and their applications, we omit a formal related work section.

2 Preliminaries

Notation. The set of all binary strings of length n is denoted as $\{0,1\}^n$, and the set of all finite binary strings as $\{0,1\}^*$. $[n]$ is the set of integers $\{1,\ldots,n\}$, and $2^{[n]}$ is the corresponding power set. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \xleftarrow{\$} X$ to represent an element x being sampled uniformly at random from a set X. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. Given a sequence $\mathbf{v}$ of n elements, we refer to its i^{th} element

[1] Note that in this setting we allow nodes to fail as long as they come back up in a bounded amount of time.

as v_i or $\mathbf{v}[i]$. If S is a set then $|S|$ refers to its cardinality. If s is a string then $|s|_2$ refers to its bit length. We denote by $\mathsf{Ber}(p)$ the Bernoulli distribution with parameter p.

Dictionaries. A dictionary structure DX of capacity n holds a collection of n label/value pairs $\{(\ell_i, v_i)\}_{i \leq n}$ and supports get and put operations. We write $v_i := \mathsf{DX}[\ell_i]$ to denote getting the value associated with label ℓ_i and $\mathsf{DX}[\ell_i] := v_i$ to denote the operation of associating the value v_i in DX with label ℓ_i. A multi-map structure MM with capacity n is a collection of n label/tuple pairs $\{(\ell_i, \mathbf{v}_i)\}_{i \leq n}$ that supports get and put operations. Similar to dictionaries, we write $\mathbf{v}_i := \mathsf{MM}[\ell_i]$ to denote getting the tuple associated with label ℓ_i and $\mathsf{MM}[\ell_i] := \mathbf{v}_i$ to denote operation of associating the tuple $\mathbf{v}_i$ to label ℓ_i.

Views. The view of a node N that participates in the execution of a randomized experiment **Exp** consists of its random coins and all messages that it sends and receives. This is a random variable which we denote by $\mathsf{view}_{\mathbf{Exp}}(N)$. When the experiment is clear from context we omit the subscript for visual clarity. We sometimes consider the joint random variable consisting of the views of multiple nodes. If S is a set of nodes, we denote by $\mathsf{view}_{\mathbf{Exp}}(S)$ the joint random variable $\langle \mathsf{view}_{\mathbf{Exp}}(N) \rangle_{N \in S}$.

3 Distributed Hash Tables

A distributed hash table is a distributed storage system that instantiates a dictionary data structure. It is distributed in the sense that the data is stored by a set of n nodes $N_1, \ldots, N_n$ and it instantiates a dictionary in the sense that it stores label/value pairs and supports Get and Put operations. Because they are distributed, DHTs rely on an overlay network which, intuitively, consists of a set of node addresses and a distributed routing protocol. As discussed in Section 1, DHTs are a fundamental primitive in distributed systems and have many applications.

In this work, we will consider two kinds of DHTs: perpetual and transient. Perpetual DHTs are composed of a fixed set of nodes that are all known at setup time. They can handle nodes going down (e.g., due to failure) and coming back online but such unresponsive nodes are expected to come back online after some period of time. Transient DHTs, on the other hand, are designed for settings where nodes are not known a-priori and can join and leave at any time. Perpetual DHTs are suitable for "permissioned" settings like the backend infrastructure of large companies whereas transient DHTs are better suited to "permissionless" settings like peer-to-peer networks and permissionless blockchains.

3.1 Perpetual DHTs

Syntax. We formalize DHTs as a collection of five algorithms $\mathsf{DHT} = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$. The first two algorithms $\mathsf{Overlay}$ and Alloc are executed only once by the entity responsible for setting up the network. $\mathsf{Overlay}$ takes as input an integer $n \geq 1$ and outputs a parameter ω from a space Ω. Alloc takes as input a parameter ω and n and outputs a parameter ψ from a space Ψ. The two parameters are used to define the overlay network and how labels are allocated to nodes. Specifically, ω determines a mapping between names and addresses and ψ determines a mapping between labels and addresses. The third algorithm, Daemon, takes ω, ψ and n as input and is executed by every node in the network. Daemon is halted only when a node wishes to leave the network and it is responsible for setting up its calling node's state, for routing messages and for storing and retrieving label/value pairs from the node's local storage. The fourth algorithm, Put,

is executed by a client to store a label/value pair on the network. Put takes as input ω and ψ and a label/value pair ℓ and v. The fifth algorithm, Get, is executed by a client to retrieve the value associated to a given label from the network. Get takes as input ω, ψ and a label ℓ and outputs a value v. Since all DHT algorithms take ω and ψ as inputs we sometimes omit them for visual clarity.

Abstracting DHTs. To instantiate a DHT, the parameters ω and ψ must be chosen together with a subset $\mathbf{C} \subseteq \mathbf{N}$ of active nodes (i.e., the nodes currently in the network) and an active set of labels $\mathbf{K} \subseteq \mathbf{L}$ (i.e., the labels stored in the DHT). Once a DHT is instantiated, we can describe it using a tuple of function families $(\mathsf{addr}, \mathsf{server}, \mathsf{route})$ that are all parameterized by ω and/or ψ. These functions are defined as

$$\mathsf{addr}_{\omega} : \mathbf{N} \to \mathbf{A} \qquad \mathsf{server}_{\omega,\psi} : \mathbf{L} \to \mathbf{A} \qquad \mathsf{route}_{\omega} : \mathbf{A} \times \mathbf{A} \to 2^{\mathbf{A}},$$

where addr_{ω} maps names from a name space $\mathbf{N}$ to addresses from an address space $\mathbf{A}$, $\mathsf{server}_{\omega,\psi}$ maps labels from a label space $\mathbf{L}$ to the address of the node that stores it, and route_{ω} maps two addresses to the addresses of the nodes on the route between them. For visual clarity we abuse notation and represent the path between two addresses by a *set* of addresses instead of as a sequence of addresses, but we stress that paths are sequences. Note that this is an abstract representation of a DHT that will be particularly useful for our analysis but, in practice, the overlay network, including its addressing and routing functions, are implemented by the Daemon algorithm.

We sometimes refer to a pair $(\omega, \mathbf{C})$ as an overlay and to a pair $(\psi, \mathbf{K})$ as an allocation. Abstractly speaking, we can think of an overlay as an assignment from active nodes to addresses and of an allocation as an assignment of active labels to addresses. In this sense, overlays and allocations are determined by a pair $(\omega, \mathbf{C})$ and $(\psi, \mathbf{K})$, respectively.

Visible addresses. A very useful notion for our purposes will be that of *visible addresses*. We say that an address $a \in \mathbf{A}$ is visible to a node $N \in \mathbf{C}$ if either: (1) there exists a label $\ell \in \mathbf{L}$ such that if ψ allocates ℓ to a then $\mathsf{server}_{\omega,\psi}(\ell) = \mathsf{addr}_{\omega}(N)$; or (2) $N \in \mathsf{route}_{\omega}(s, a)$. The intuition behind this is that if a label ℓ is mapped to an address in $\mathsf{Vis}(s, N)$ then N either stores the label ℓ or routes it. Notice that the set of visible addresses also depend on parameter ω and the set $\mathbf{C}$ of nodes that are currently active. We therefore subscript $\mathsf{Vis}_{\omega,\mathbf{C}}(s, N)$ with the overlay $(\omega, \mathbf{C})$. We also extend the notion to the set of visible addresses $\mathsf{Vis}_{\omega,\mathbf{C}}(s, S)$ for a set of nodes $S \subseteq \mathbf{C}$ which is defined simply as $\mathsf{Vis}_{\omega,\mathbf{C}}(s, S) = \cup_{N \in S} \mathsf{Vis}_{\omega,\mathbf{C}}(s, N)$.

Allocation distribution. Another important notion in our analysis is what we refer to as a label's *allocation distribution* which is the probability distribution that governs the address at which a label is allocated. More precisely, this is captured by the random variable $\psi(\ell)$, where ψ is sampled by the algorithm Alloc. In this work, we assume allocation distributions are *label-independent* in the sense that every label's allocation distribution is the same [2]. We therefore simply refer to this distribution as the DHT's allocation distribution.

Given a DHT's allocation distribution, we also consider a distribution $\Delta_{\omega,\mathbf{C}}(S)$ that is parameterized by a set of addresses $S \subseteq \mathbf{A}$. This distribution is over S and has probability mass function

$$f_{\Delta(S)}(a) = \frac{f_{\psi}(a)}{\sum_{a \in S} f_{\psi}(a)} = \frac{\Pr\left[\,\psi(\ell) = a\,\right]}{\Pr\left[\,\psi(\ell) \in S\,\right]},$$

where f_{ψ} is the probability mass function of the DHT's allocation distribution.

[2] This is true for every DHT we are aware of [20, 14, 27, 12].

Non-committing allocations. As we will see in Section 4.2, our EDHT construction can be based on any DHT but the security of the resulting scheme will depend on certain properties of the underlying DHT. We describe these properties here. The first property that we require of a DHT is that the overlays it produces be non-committing in the sense that it supports a form of equivocation with respect to its allocation. More precisely, for some fixed overlay $(\omega, \mathbf{C})$ and allocation $(\psi, \mathbf{K})$, there should exist some efficient mechanism to arbitrarily change/program ψ. In other words, there should exist a polynomial-time algorithm $\mathsf{Program}$ such that, for all $(\omega, \mathbf{C})$ and $(\psi, \mathbf{K})$, given a label $\ell \in \mathbf{L}$ and address $a \in \mathbf{A}$, $\mathsf{Program}(\ell, a)$ modifies the DHT so that $\psi(\ell) = a$ (in a manner that is indistinguishable to the get and put operations). For the special case of Chord, which we study in Section 5, this can be achieved by modeling one of its hash functions as a random oracle.

Balanced overlays. The second property is related to how well the DHT load balances the label/value pairs it stores. While load balancing is clearly important for storage efficiency we will see, perhaps surprisingly, that it also has an impact on security. Intuitively, we say that an overlay $(\omega, \mathbf{C})$ is balanced if for all labels ℓ, the probability that any set of θ nodes sees ℓ is not too large.

Definition 3.1 (Balanced overlays)**.** *Let $\omega \in \Omega$ be an overlay parameter and let $\mathbf{C} \subseteq \mathbf{N}$ be a set of active nodes. We say that an overlay $(\omega, \mathbf{C})$ is (ε, θ)-balanced if for all $\ell \in \mathbf{L}$, for all $s \in \mathbf{A}$ and for all $S \subseteq \mathbf{C}$ with $|S| = \theta$,*

$$\Pr\left[\, \mathsf{server}_{\omega,\psi}(\ell) \in \mathsf{Vis}_{\omega,\mathbf{C}}(s, S) \,\right] \leq \varepsilon,$$

where the probability is over the coins of Alloc *and where ε can depend on θ.*

Definition 3.2 (Balanced DHT)**.** *We say that a distributed hash table* $\mathsf{DHT} = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$ *is $(\varepsilon, \delta, \theta)$-balanced if for all $\mathbf{C} \subseteq \mathbf{N}$, the probability that an overlay $(\omega, \mathbf{C})$ is (ε, θ)-balanced is at least $1 - \delta$ over the coins of* $\mathsf{Overlay}$ *and where ε and δ can depend on $\mathbf{C}$ and θ.*

3.2 Transient Distributed Hash Tables

In this section, we formalize DHTs in the context of transient networks.

Syntax. Transient DHTs are a collection of seven algorithms $\mathsf{DHT}^+ = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get}, \mathsf{Leave}, \mathsf{Join})$. The first five algorithms are same as in the perpetual setting. The sixth is an algorithm Leave executed by a node $N \in \mathbf{C}$ when it wishes to leave the network. Leave takes nothing as input and outputs nothing but it halts the Daemon algorithm. The seventh is an algorithm Join that is executed by a node $N \in \mathbf{N} \setminus \mathbf{C}$ that wishes to join the network. It takes nothing as input and outputs nothing but executes the Daemon algorithm. When a node executes a Leave or Join, the routing tables of all the other nodes are updated and label/value pairs are moved around in the network according to allocation ψ. In other words, when a node leaves, its pairs are reallocated in the network and when a node joins, some pairs stored on the other nodes are moved to the new node.

Note that when a node $N \in \mathbf{C}$ leaves the network, the set of active nodes $\mathbf{C}$ automatically shrinks to exclude N. Similarly, when a node $N \in \mathbf{N} \setminus \mathbf{C}$ joins the network, the set of active nodes $\mathbf{C}$ expands to include N. From now on, whenever we write $\mathbf{C}$ we are referring to the current set of active nodes.

Stability. To prove the security of EDHTs in the transient setting, we need the underlying DHT to satisfy a stronger notion than balance which we call *stability*.

Definition 3.3 (Stability)**.** *We say that a transient distributed hash table* $\mathsf{DHT}^+ = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get}, \mathsf{Leave}, \mathsf{Join})$ *is* $(\varepsilon, \delta, \theta)$*-stable if*

$$\Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta} (\omega, \mathbf{C}) \text{ is } (\varepsilon,\theta)\text{-balanced}\right] \geq 1-\delta$$

where the probability is over the choice of ω*, and* $\varepsilon = \varepsilon(\mathbf{C})$*.*

Notice that stability requires that $\mathsf{Overlay}$ returns an overlay parameter ω such that, with high probability, $(\omega, \mathbf{C})$ is balanced *for all* possible subsets of active nodes $\mathbf{C}$ simultaneously. Balance, on the other hand, only requires that for all sets of active nodes $\mathbf{C}$, with high probability $\mathsf{Overlay}$ will output an overlay parameter ω such that $(\omega, \mathbf{C})$ is balanced. In other words, stability requires a single overlay parameter ω that is "good" for all subsets of active nodes whereas balance does not.

4 Encrypted Distributed Hash Tables in the Perpetual Setting

In this Section, we formally define encrypted distributed hash tables. An EDHT is an end-to-end encrypted distributed system that instantiates a dictionary data structure.

4.1 Syntax and Security Definitions

Syntax. We formalize symmetric EDHTs as a collection of six algorithms $\mathsf{EDHT} = (\mathsf{Gen}, \mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$. The first algorithm Gen is executed by a client and takes as input a security parameter 1^k and outputs a secret key K. $\mathsf{Overlay}$ and Alloc are executed only once by the entity responsible for setting up the network. $\mathsf{Overlay}$ takes as input an integer $n \geq 1$ and output a parameter $\omega \in \Omega$. Alloc takes as input ω and n and outputs a parameter $\psi \in \Psi$. The fourth algorithm, Daemon, takes ω, ψ and n as input and is executed by every node in the network. Daemon is halted only when a node wishes to leave the network. The fifth algorithm, Put, is executed by a client to store a label/value pair on the network. Put takes as input the secret key K and a label/value pair (ℓ, v). The sixth algorithm, Get, is executed by a client to retrieve the value associated to a given label from the network. Get takes as input the secret key K and a label ℓ and outputs a value v.

Security. We now turn to formalizing the security of an EDHT. We do this by combining the definitional approaches used in secure multi-party computation [5] and in structured encryption [9, 8]. The security of multi-party protocols is generally formalized using the Real/Ideal-world paradigm. This approach consists of defining two probabilistic experiments **Real** and **Ideal** where the former represents a real-world execution of the protocol where the parties are in the presence of an adversary, and the latter represents an ideal-world execution where the parties interact with a trusted functionality. The protocol is secure if no environment can distinguish between the outputs of these two experiments. Below, we will describe both these experiments more formally.

Before doing so, we discuss a minor extension to the standard definitions. To capture the fact that a protocol could leak information to the adversary, we parameterize the definition with a leakage profile that consists of a leakage function $\mathcal{L}$ that captures the information leaked by the Put and Get operations. Our motivation for making the leakage explicit is to highlight its presence.

Functionality $\mathcal{F}_{\mathsf{EDHT}}^{\mathcal{L}}$

$\mathcal{F}_{\mathsf{EDHT}}^{\mathcal{L}}$ stores a dictionary DX initialized to empty and proceeds as follows, running with client $\mathcal{C}$, n nodes $N_1, \ldots, N_n$ and a simulator Sim:

- $\mathsf{Put}(\ell, v)$: Upon receiving a label/value pair (ℓ, v) from client $\mathcal{C}$, it sets $\mathsf{DX}[\ell] := v$, and sends the leakage $\mathcal{L}(\mathsf{DX}, (\mathtt{put}, \ell, v))$ to the simulator Sim.
- $\mathsf{Get}(\ell)$: Upon receiving a label ℓ from client C, it returns $\mathsf{DX}[\ell]$ to the client $\mathcal{C}$ and the leakage $\mathcal{L}(\mathsf{DX}, (\mathtt{get}, \ell, \perp))$ to the simulator Sim.

Figure 1: $\mathcal{F}_{\mathsf{EDHT}}^{\mathcal{L}}$: The EDHT functionality parameterized with leakage function $\mathcal{L}$.

The real-world experiment. The experiment is executed between a trusted party $\mathcal{T}$, a client $\mathcal{C}$, a set $\mathbf{C} \subseteq \mathbf{N}$ of n nodes $N_1, \ldots, N_n$, an environment $\mathcal{Z}$ and an adversary $\mathcal{A}$. The trusted party $\mathcal{T}$ runs $\mathsf{Overlay}(n)$ and $\mathsf{Alloc}(\omega, n)$ and sends (ω, ψ) to all parties, i.e., the nodes, the client, the environment and the adversary. Given $z \in \{0,1\}^*$, the environment $\mathcal{Z}$ sends to the adversary $\mathcal{A}$, a subset $I \subseteq \mathbf{C}$ of nodes to corrupt. The client $\mathcal{C}$ generates a secret key $K \leftarrow \mathsf{Gen}(1^k)$. $\mathcal{Z}$ then adaptively chooses a polynomial number of operations op_j, where $\mathsf{op}_j \in \{\mathtt{get}, \mathtt{put}\} \times \mathbf{L} \times \{\mathbf{V}, \perp\}$ and sends it to $\mathcal{C}$. If $\mathsf{op}_j = (\mathtt{get}, \ell)$, the client $\mathcal{C}$ executes $\mathsf{EDHT.Get}(K, \ell)$. If $\mathsf{op}_j = (\mathtt{put}, \ell, v)$, $\mathcal{C}$ initiates $\mathsf{EDHT.Put}(K, \ell, v)$. The client forwards its output from running the get/put operations to $\mathcal{Z}$. $\mathcal{A}$ computes a message m from its view and sends it to $\mathcal{Z}$. Finally, $\mathcal{Z}$ returns a bit that is output by the experiment. We let $\mathbf{Real}_{\mathcal{A},\mathcal{Z}}(k)$ be a random variable denoting $\mathcal{Z}$'s output bit.

The ideal-world experiment. The experiment is executed between a client $\mathcal{C}$, a set $\mathbf{C} \subseteq \mathbf{N}$ of n nodes $N_1, \ldots, N_n$, an environment $\mathcal{Z}$ and a simulator Sim. Each party also has access to the ideal functionality $\mathcal{F}_{\mathsf{EDHT}}^{\mathcal{L}}$. Given $z \in \{0,1\}^*$, the environment $\mathcal{Z}$ sends to the simulator Sim, a subset $I \subseteq \mathbf{C}$ of nodes to corrupt. $\mathcal{Z}$ then adaptively chooses a polynomial number of operations op_j, where $\mathsf{op}_j \in \{\mathtt{get}, \mathtt{put}\} \times \mathbf{L} \times \{\mathbf{V}, \perp\}$, and sends it to the client $\mathcal{C}$ which, in turn, forwards it to $\mathcal{F}_{\mathsf{EDHT}}^{\mathcal{L}}$. If $\mathsf{op}_j = (\mathtt{get}, \ell)$, the functionality executes $\mathcal{F}_{\mathsf{EDHT}}^{\mathcal{L}}.\mathsf{Get}(\ell)$. Otherwise, if $\mathsf{op}_j = (\mathtt{put}, \ell, v)$ the functionality executes $\mathcal{F}_{\mathsf{EDHT}}^{\mathcal{L}}.\mathsf{Put}(\ell, v)$. $\mathcal{C}$ forwards its outputs to $\mathcal{Z}$ whereas Sim sends $\mathcal{Z}$ some arbitrary message m. Finally, $\mathcal{Z}$ returns a bit that is output by the experiment. We let $\mathbf{Ideal}_{\mathsf{Sim},\mathcal{Z}}(k)$ be a random variable denoting $\mathcal{Z}$'s output bit.

Definition 4.1 ($\mathcal{L}$-security)**.** *We say that an encrypted distributed hash table* $\mathsf{EDHT} = (\mathsf{Gen}, \mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$ *is $\mathcal{L}$-secure, if for all* PPT *adversaries $\mathcal{A}$ and all* PPT *environments $\mathcal{Z}$, there exists a* PPT *simulator* Sim *such that for all $z \in \{0,1\}^*$,*

$$|\Pr[\mathbf{Real}_{\mathcal{A},\mathcal{Z}}(k) = 1] - \Pr[\mathbf{Ideal}_{\mathsf{Sim},\mathcal{Z}}(k) = 1]| \leq \mathsf{negl}(k).$$

4.2 The Standard EDHT in the Perpetual Setting

We now describe the standard approach to storing sensitive data on a DHT. This approach relies on simple cryptographic primitives and a non-committing and balanced DHT.

Overview. The scheme $\mathsf{EDHT} = (\mathsf{Gen}, \mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$ is described in detail in Figure 2 and, at a high level, works as follows. It makes black-box use of a distributed hash table $\mathsf{DHT} = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$, a pseudo-random function F and a symmetric-key encryption scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

Let $\mathsf{DHT} = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$ be a distributed hash table, $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric-key encryption scheme and F be a pseudo-random function. Consider the encrypted distributed hash table $\mathsf{EDHT} = (\mathsf{Gen}, \mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$ that works as follows:

- $\mathsf{Gen}(1^k)$:
  1. sample $K_1 \xleftarrow{\$} \{0,1\}^k$ and compute $K_2 \leftarrow \mathsf{SKE.Gen}(1^k)$
  2. output $K = (K_1, K_2)$
- $\mathsf{Overlay}(n)$:
  1. compute and output $\omega \leftarrow \mathsf{DHT.Overlay}(n)$
- $\mathsf{Alloc}(n, \omega)$:
  1. compute and output $\psi \leftarrow \mathsf{DHT.Alloc}(n, \omega)$
- $\mathsf{Daemon}(\omega, \psi, n)$:
  1. Execute $\mathsf{DHT.Daemon}(\omega, \psi, n)$
- $\mathsf{Put}(K, \ell, v)$:
  1. Parse K as (K_1, K_2)
  2. compute $t := F_{K_1}(\ell)$
  3. compute $e \leftarrow \mathsf{SKE.Enc}(K_2, v)$
  4. execute $\mathsf{DHT.Put}(t, e)$
- $\mathsf{Get}(K, \ell)$:
  1. Parse K as (K_1, K_2)
  2. Initialise $v := \perp$
  3. compute $t := F_{K_1}(\ell)$
  4. execute $e \leftarrow \mathsf{DHT.Get}(t)$
  5. if $e \neq \perp$, compute and output $v \leftarrow \mathsf{SKE.Dec}(K_2, e)$

Figure 2: EDHT: An Encrypted Distributed Hash Table

The Gen algorithm takes as input a security parameter 1^k and uses it to generate a key K_1 for the pseudo-random function F and a key K_2 for the symmetric encryption scheme SKE. It then outputs a key $K = (K_1, K_2)$. The Overlay algorithm takes as input an integer $n \geq 1$, and generates and outputs ω by executing $\mathsf{DHT.Overlay}(n)$. Similarly, the Alloc algorithm takes as input n and ω and generates and outputs a parameter ψ by executing $\mathsf{DHT.Alloc}(n, \omega)$. The Daemon algorithm takes as input ω, ψ and n and executes $\mathsf{DHT.Daemon}(\omega, \psi, n)$. The Put algorithm takes as input the secret key K and a label/value pair (ℓ, v). It first computes $t := F_{K_1}(\ell)$ and $e \leftarrow \mathsf{Enc}(K_2, v)$ and then executes $\mathsf{DHT.Put}(t, e)$. The Get algorithm takes as input the secret key K and a label ℓ. It computes $t := F_{K_1}(\ell)$ and executes $e \leftarrow \mathsf{DHT.Get}(t)$. It then outputs $\mathsf{SKE.Dec}(K, e)$.

Security. We now describe the leakage of EDHT. Intuitively, it reveals to the adversary the times at which a label is stored or retrieved with some probability. More formally, it is defined with the following *stateful* leakage function

- $\mathcal{L}_\varepsilon(\mathsf{DX}, (\mathsf{op}, \ell, v))$:

1. if ℓ has never been seen
   (a) sample and store $b_\ell \leftarrow \mathsf{Ber}(\varepsilon)$
2. if $b_\ell = 1$
   (a) if $\mathsf{op} = \mathtt{put}$ output $(\mathtt{put}, \mathsf{qeq}(\ell))$
   (b) else if $\mathsf{op} = \mathtt{get}$ output $(\mathtt{get}, \mathsf{qeq}(\ell))$
3. else if $b_\ell = 0$
   (a) output $\perp$

where qeq is the *query equality pattern* which reveals if and when a label was queried or put in the past. Note that when $\varepsilon = 1$ (for some θ), $\mathcal{L}_\varepsilon$ reduces to the leakage profile achieved by standard encrypted dictionary constructions [8, 6]. On the other hand, when $\varepsilon < 1$, this leakage profile is "better" than the profile of known constructions.

Discussion. We now explain why the leakage function is probabilistic and why it depends on the balance of the underlying DHT. Intuitively, one expects that the adversary's view is only affected by get and put operations on labels that are either: (1) allocated to a corrupted node; or (2) allocated to an uncorrupted node whose path (starting from the client) includes a corrupted node. In such a case, the adversary's view would not be affected by all operations but only a subset of them. Our leakage function captures this intuition precisely and it is probabilistic because, in the real world, the subset of operations that affect the adversary's view is determined probabilistically because it depends on the choice of overlay and allocation—both of which are chosen at random. The way this is handled in the leakage function is by sampling a bit b with some probability and revealing leakage on the current operation if $b = 1$. This determines the subset of operations whose leakage will be visible to the adversary.

Now, for the simulation to go through, the operations simulated by the simulator need to be visible to the adversary with the same probability as in the real execution. But these probabilities depend on ω and ψ which are not known to the leakage function. Note that this implies a rather strong definition in the sense that the scheme hides information about the overlay and the allocation of the DHT.

Since ω and ψ are unknown to the leakage function, the leakage function can only guess as to what they could be. But because the DHT is guaranteed to be $(\varepsilon, \delta, \theta)$-balanced, the leakage function can assume that, with probability at least $1 - \delta$, the overlay will be (ε, θ)-balanced which, in turn, guarantees that the probability that a label is visible to any adversary with at most θ corruptions is at most ε. Therefore, in our leakage function, we can set the probability that $b = 1$ to be ε in the hope that simulator can "adjust" the probability internally to be in accordance to the ω that it sampled. Note that the simulator can adjust the probability only if for its own chosen ω, the probability that a query is visible to the adversary is less than ε. But this will happen with probability at least $1 - \delta$ so the simulation will work with probability at least $1 - \delta$.

We are now ready to state our main security Theorem which proves that the standard EDHT construction is $\mathcal{L}_\varepsilon$-secure with probability that is negligibly close to $1 - \delta$ when its underlying DHT is $(\varepsilon, \delta, \theta)$-balanced.

Theorem 4.2. *If $|I| \leq \theta$ and if* DHT *is $(\varepsilon, \delta, \theta)$-balanced and has non-committing and label-independent allocations, then* EDHT *is $\mathcal{L}_\varepsilon$-secure with probability at least $1 - \delta - \mathsf{negl}(k)$.*

Proof. Consider the simulator Sim that works as follows. Given a set of corrupted nodes $I \subseteq \mathbf{C}$, it computes $\omega \leftarrow \mathsf{DHT.Overlay}(n)$, initializes n nodes $N_1, \ldots, N_n$ in $\mathbf{C}$, simulates the adversary $\mathcal{A}$ with

I as input and generates a symmetric key $K \leftarrow \mathsf{SKE.Gen}(1^k)$. In the following, let $B \stackrel{def}{=} \mathsf{Vis}_{\omega,\mathbf{C}}(s, I)$ and $p' \stackrel{def}{=} \Pr[\,\psi(\ell) \in B\,]$, which is unique since we assume label-independent allocations. If $p' > \varepsilon$, the simulator aborts otherwise it continues.

When a put/get operation is executed, Sim receives from $\mathcal{F}_{\mathsf{EDHT}}$ the leakage

$$\lambda \in \left\{ \left(\mathtt{put}, \mathsf{qeq}(\ell) \right), \left(\mathtt{get}, \mathsf{qeq}(\ell) \right), \perp \right\}.$$

If $\lambda = \perp$ then Sim does nothing. If $\lambda \neq \perp$, then Sim checks the query equality to see if the label has been used in the past. If not, it samples and stores a bit

$$b' \leftarrow \mathsf{Ber}\left(\frac{p'}{\varepsilon}\right).$$

Note that, this is indeed a valid Bernoulli distribution since

$$p' = \Pr[\,\psi(\ell) \in B\,] = \Pr[\,\mathsf{server}_{\omega,\psi}(\ell) \in \mathsf{Vis}_{\omega,\mathbf{C}}(s, I)\,] \leq \varepsilon,$$

where the second equality follows from the definition of visible address, and the last inequality follows from $|I| \leq \theta$ and $(\omega, \mathbf{C})$ being (ε, θ)-balanced.

If the label was seen in the past, Sim retrieves the bit b' that was previously sampled. If $b' = 0$, then it does nothing, but if $b' = 1$ it uses the query equality to check if the label has been used in the past. If so, it sets t to the d-bit value previously used. If not, it sets $t \stackrel{\$}{\leftarrow} \{0,1\}^d$, computes $e \leftarrow \mathsf{SKE.Enc}(K, 0)$, and samples an address $a \leftarrow \Delta_{\omega,\mathbf{C}}(B)$, and programs ψ to map t to a. Finally, if the operation was a put, it executes $\mathsf{DHT.Put}(t, e)$, otherwise it executes $\mathsf{DHT.Get}(t)$. Once all of the environment's operations are processed, the simulator returns whatever the adversary outputs.

It remains to show that the view of the adversary $\mathcal{A}$ during the simulation is indistinguishable from its view in a **Real** experiment. We do this using a sequence of games.

Game_0 : is the same as a $\mathbf{Real}_{\mathcal{A},\mathcal{Z}}(k)$ experiment.

Game_1 : is the same as Game_0 except that the encryption of the value v during a Put is replaced by $\mathsf{SKE.Enc}(K_2, 0)$.

Game_2 : is the same as Game_1 except that output of the PRF F is replaced by a truly random string of d bits.

Game_3 : is the same as Game_2 except that for each operation (op, ℓ, v) (where v can be null), we check if ℓ has been seen before. If not, we sample a bit $b_\ell \leftarrow \mathsf{Ber}(\varepsilon)$, else we set b_ℓ to the bit previously sampled. If $b_\ell = 1$ and $\mathsf{op} = (\mathtt{put}, \ell, v)$, we replace the Put operation with $\mathsf{Sim}(\mathtt{put}, \mathsf{qeq}(\ell))$, and if $b_\ell = 1$ and $\mathsf{op} = (\mathtt{get}, \ell)$, we replace the Get operation with $\mathsf{Sim}(\mathtt{get}, \mathsf{qeq}(\ell))$. If $b_\ell = 0$, we do nothing.

Game_1 is indistinguishable from Game_0, otherwise the encryption scheme is not semantically secure. Game_2 is indistinguishable from Game_1 because the outputs of pseudorandom functions are indistinguishable from random strings.

We now show that the adversary's views in Game_2 and Game_3 are indistinguishable. We denote

these views by $\mathbf{view_2}(I)$ and $\mathbf{view_3}(I)$, respectively, and consider the ith "sub-views" $\mathbf{view_2}^i(I)$ and $\mathbf{view_3}^i(I)$ which include the set of messages seen by the adversary (through the corrupted nodes) during the execution of op_i. Let $\mathbf{op}$ denote the sequence of q operations generated by the environment. Let $\ell_1, \ldots, \ell_q$ be the labels of the operations in $\mathbf{op}$, and let $t_1, \ldots, t_q$ be the corresponding random strings obtained by replacing $F_K(\ell_i)$ with random strings. Because DHT is $(\varepsilon, \delta, \theta)$-balanced, we know that with probability at least $1 - \delta$, the overlay $(\omega, \mathbf{C})$ will be (ε, θ)-balanced. So for the remainder of the proof, we assume the overlay is (ε, θ)-balanced.

First, we treat the case where t_i (or equivalently ℓ_i) has never been seen before. Let $\mathcal{E}_i$ be the event that $\psi(t_i) \in B$, where $B = \mathsf{Vis}_{\omega,\mathbf{C}}(s, I)$ are the addresses visible to the corrupted nodes. For all possible views $\mathbf{v}$, we have

$$\begin{aligned}
&\Pr\left[\,\mathbf{view_2}^i(I) = \mathbf{v}\,\right] \\
&\quad = \Pr\left[\,\mathbf{view_2}^i(I) = \mathbf{v} \wedge \mathcal{E}_i\,\right] + \Pr\left[\,\mathbf{view_2}^i(I) = \mathbf{v} \wedge \overline{\mathcal{E}_i}\,\right] \\
&\quad = \Pr\left[\,\mathbf{view_2}^i(I) = \mathbf{v} \mid \mathcal{E}_i\,\right] \cdot \Pr\left[\,\mathcal{E}_i\,\right] + \Pr\left[\,\mathbf{view_2}^i(I) = \mathbf{v} \mid \overline{\mathcal{E}_i}\,\right] \cdot \Big(1 - \Pr\left[\,\mathcal{E}_i\,\right]\Big) \\
&\quad = \Pr\left[\,\mathbf{view_2}^i(I) = \mathbf{v} \mid \mathcal{E}_i\,\right] \cdot \Pr\left[\,\mathcal{E}_i\,\right]
\end{aligned}$$

where the third equality follows from the fact that, conditioned on $\overline{\mathcal{E}_i}$, the nodes in I do not see any messages at all.

Turning to $\mathbf{view_3}$, let $\mathcal{Q}_i$ be the event that $b_i = 1 \wedge b'_i = 1$. Then for all possible views $\mathbf{v}$, we have

$$\begin{aligned}
&\Pr\left[\,\mathbf{view_3}^i(I) = \mathbf{v}\,\right] \\
&\quad = \Pr\left[\,\mathbf{view_3}^i(I) = \mathbf{v} \wedge \mathcal{Q}_i\,\right] + \Pr\left[\,\mathbf{view_3}^i(I) = \mathbf{v} \wedge \overline{\mathcal{Q}_i}\,\right] \\
&\quad = \Pr\left[\,\mathbf{view_3}^i(I) = \mathbf{v} \mid \mathcal{Q}_i\,\right] \cdot \Pr\left[\,\mathcal{Q}_i\,\right] + \Pr\left[\,\mathbf{view_2}^i(I) = \mathbf{v} \mid \overline{\mathcal{Q}_i}\,\right] \cdot \Big(1 - \Pr\left[\,\mathcal{Q}_i\,\right]\Big) \\
&\quad = \Pr\left[\,\mathbf{view_3}^i(I) = \mathbf{v} \mid \mathcal{Q}_i\,\right] \cdot \Pr\left[\,\mathcal{Q}_i\,\right] \qquad (1)
\end{aligned}$$

where the third equality follows from the fact that, for all i, conditioned on $\overline{\mathcal{Q}_i}$, either Sim is never executed or Sim does nothing. In either case, the nodes in I will not see any messages, so for all $\mathbf{v}$ we have $\Pr\left[\,\mathbf{view_3}^i(I) = \mathbf{v} \mid \overline{\mathcal{Q}_i}\,\right] = 0$.

Notice, however, that

$$\Pr\left[\,\mathcal{Q}_i\,\right] = \Pr\left[\,b_i = 1 \wedge b'_i = 1\,\right] = \varepsilon \cdot \frac{\Pr\left[\,\psi(t_i) \in B\,\right]}{\varepsilon} = \Pr\left[\,\psi(t_i) \in B\,\right] = \Pr\left[\,\mathcal{E}_i\,\right],$$

so to show that the views are equally distributed it remains to show that for all $\mathbf{v}$,

$$\Pr\left[\,\mathbf{view_2}^i(I) = \mathbf{v} \mid \mathcal{E}_i\,\right] = \Pr\left[\,\mathbf{view_3}^i(I) = \mathbf{v} \mid \mathcal{Q}_i\,\right]. \qquad (2)$$

To see why this holds, notice that, conditioned on $\mathcal{E}_i$ and $\mathcal{Q}_i$, the only difference between Game_2 and Game_3 is that, in the former, the labels t_i are mapped to an address a according to an allcoation $(\psi, \mathbf{K})$ generated using Alloc, whereas in the latter, the labels t_i are programmed to an address a sampled from $\Delta_{\omega,\mathbf{C}}(B)$. We show, however, that in both cases, the labels t_i are allocated with the same probability distribution. In Game_2, for all $a \in B$, we have

$$\Pr\left[\,\psi(t_i) = a \mid \mathcal{E}_i\,\right] = \frac{\Pr\left[\,\psi(t_i) = a \wedge \mathcal{E}_i\,\right]}{\Pr\left[\,\mathcal{E}_i\,\right]} = \frac{\Pr\left[\,\psi(t_i) = a\,\right]}{\Pr\left[\,\mathcal{E}_i\,\right]} = \frac{\Pr\left[\,\psi(t_i) = a\,\right]}{\Pr\left[\,\psi(t_i) \in B\,\right]},$$

where the second equality follows from the fact that the event $\{\psi(t_i) = a\} \subseteq \mathcal{E}_i$.
In Game_3, for all $a \in B$, we have,

$$\Pr[\,\psi(t_i) = a \mid \mathcal{Q}_i\,] = \frac{\Pr[\,\psi(t_i) = a\,]}{\Pr[\,\psi(t_i) \in B\,]},$$

since a is sampled from $\Delta_{\omega,\mathbf{C}}(B)$. Since, for all i, conditioned on $\mathcal{Q}_i$ and E_i, labels are allocated to addresses with the same distribution in both games and since this is the only difference between the games,

$$\Pr\Big[\,\mathbf{view_3}^i(I) = \mathbf{v} \mid \mathcal{Q}_i\,\Big] = \Pr\Big[\,\mathbf{view_2}^i(I) = \mathbf{v} \mid \mathcal{E}_i\,\Big]. \tag{3}$$

Plugging Eq. 3 into Eq. 1, we have that for all i and all $\mathbf{v}$,

$$\Pr\Big[\,\mathbf{view_2}^i(I) = \mathbf{v}\,\Big] = \Pr\Big[\,\mathbf{view_3}^i(I) = \mathbf{v}\,\Big].$$

Now we consider the case where t_i has been seen in the past. In this case, Put or Get operations will produce the same messages that were generated in the past which means that $\mathbf{view_2}^i(I)$ will be the same as before. Similarly, $\mathbf{view_3}^i(I)$ will be the same as before because, whenever t_i has been seen in the past, Sim behaves the same.

■

Efficiency. The standard scheme does not add much to the put and get complexities of the underlying DHT. Precisely, the get complexity is

$$\mathsf{time}^{\mathsf{get}}_{\mathsf{DHT}}(\ell) + \mathsf{time}^{\mathsf{dec}}_{\mathsf{SKE}} + \mathsf{time}_{\mathsf{PRF}} = O\big(\mathsf{time}^{\mathsf{get}}_{\mathsf{DHT}}(\ell)\big),$$

where $\mathsf{time}^{\mathsf{get}}_{\mathsf{DHT}}(\ell)$ is the get complexity of the DHT, $\mathsf{time}^{\mathsf{dec}}_{\mathsf{SKE}}$ is the cost of decryption with SKE, and $\mathsf{time}_{\mathsf{PRF}}$ is the cost of a PRF evaluation with F. Similarly, the put complexity of EDHT is

$$\mathsf{time}^{\mathsf{put}}_{\mathsf{DHT}}(\ell, v) + \mathsf{time}^{\mathsf{enc}}_{\mathsf{SKE}} + \mathsf{time}_{\mathsf{PRF}} = O\big(\mathsf{time}^{\mathsf{put}}_{\mathsf{DHT}}(\ell, v)\big),$$

where $\mathsf{time}^{\mathsf{put}}_{\mathsf{DHT}}(\ell, v)$ is the put complexity of the DHT and $\mathsf{time}^{\mathsf{Enc}}_{\mathsf{SKE}}$ is the cost of encryption with SKE. The round, communication and storage complexities of the scheme are the same as the underlying DHT.

5 A Chord-Based EDHT in the Perpetual Setting

In this section, we analyze the security of the standard EDHT when its underlying DHT is instantiated with Chord. We first give a brief overview of how Chord works and then show that: (1) it has non-committing overlays in the random oracle model; and (2) it is balanced.

Setting up Chord. For Chord, the space Ω is the set of all hash functions $\mathcal{H}_1$ from $\mathbf{N}$ to $\mathbf{A} = \{0, \ldots, 2^m - 1\}$. $\mathsf{Overlay}$ samples a hash function H_1 uniformly at random from $\mathcal{H}_1$ and outputs $\omega = H_1$. The map addr_ω is the hash function itself so Chord assigns to each active node $N \in \mathbf{C}$ an address $H_1(N)$ in $\mathbf{A}$. We call the set $\chi_{\mathbf{C}} = \{H_1(N_1), \ldots, H_1(N_n)\}$ of addresses assigned to active nodes a *configuration*.

The parameter space Ψ is the set of all hash functions $\mathcal{H}_2$ from $\mathbf{L}$ to $\mathbf{A} = \{0, \ldots, 2^m - 1\}$. Alloc samples a hash function H_2 uniformly at random from $\mathcal{H}_2$ and outputs $\psi = H_2$. The map $\mathsf{server}_{\omega,\psi}$

maps every label ℓ in $\mathbf{L}$ to the address of the active node that is closest to $H_2(\ell)$ (in a clockwise direction). More formally, $\mathsf{server}_{\omega,\psi}$ is the function $\mathsf{succ}_{\chi_\mathbf{C}} \circ H_2$, where $\mathsf{succ}_{\chi_\mathbf{C}}$ is the *successor* function that assigns each address in $\mathbf{A}$ to its least upper bound in $\chi_\mathbf{C}$. Here, $\{0, \ldots, 2^m - 1\}$ is viewed as a "ring" in the sense that the successor of 2^{m-1} is 0.

Based on $\omega = H_1$, the Daemon algorithm constructs a routing table by storing the addresses of the node's 2^ith successor where $0 \leq i \leq \log n$ (we refer the reader to [27] for more details). Note that a routing table contains at most $\log n$ other nodes. The Chord routing protocol is fairly simple: given a message destined to a node N_d, a node N checks if $N = N_d$. If not, the node forwards the message to the node N' in its routing table with an address closest to N_d. Note that the route_ω map for Chord is deterministic given a fixed set of active nodes and it guarantees that any two nodes have a path of length at most $\log n$.

Storing and retrieving. Once the DHT is instantiated, each Chord node instantiates an empty dictionary data structure DX_i. When a client executes a Put operation on a label/value pair (ℓ, v), it computes $N_\ell = \mathsf{succ}_{\chi_\mathbf{C}}(H_2(\ell))$ and uses the Chord routing protocol to send the pair (ℓ, v) to the node N_ℓ who stores it in its local dictionary DX_i. When executing a Get query on a label ℓ, the Client also computes $N_\ell = \mathsf{succ}_{\chi_\mathbf{C}}(H_2(\ell))$ and, again, uses the Chord routing protocol to send the label ℓ to N_ℓ. The latter looks up ℓ in its local dictionary DX_i and uses the Chord routing protocol to return the associated value v.

Visible addresses. Given a fixed overlay $(H_1, \mathbf{C})$, an address $s \in \mathbf{A}$, and a node $N \in \mathbf{C}$,

$$\mathsf{Vis}_{\chi_\mathbf{C}}(s, N) = \left\{ \mathsf{arc}_{\chi_\mathbf{C}}(N') : N \in \mathsf{route}_{\chi_\mathbf{C}}(s, N') \right\} \bigcup \mathsf{arc}_{\chi_\mathbf{C}}(N),$$

and, for any set $S \subseteq \mathbf{C}$, $\mathsf{Vis}_{\omega,\mathbf{C}}(s, S) = \cup_{N \in S} \mathsf{Vis}_{\omega,\mathbf{C}}(s, N)$.

Non-committing allocation. Given a label ℓ and an address θ, the allocation $(H_2, \mathbf{K})$ can be changed by programming the random oracle H_2 to output θ when it is queried on ℓ.

Allocation distribution. We now describe Chord's allocation distribution. Since Chord assigns labels to addresses using a random oracle H_2, it follows that for all overlays $(H_1, \mathbf{C})$, all labels $\ell \in \mathbf{L}$ and all addresses $a \in \mathbf{A}$,

$$f_{H_2}(a) = \Pr\left[\, H_2(\ell) = a \,\right] = \frac{1}{|\mathbf{A}|},$$

which implies that Chord has label-independent allocations. From this it also follows that $\Delta_{H_1,\mathbf{C}}(S)$ has a probability mass function

$$f_{\Delta(S)}(a) = \frac{1}{|S|}.$$

5.1 Analyzing Chord's Maximum Area

As we showed in Theorem 4.2, the leakage profile of the standard EDHT depends on the balance of the underlying DHT. As we will see, analyzing the balance of Chord is non-trivial and relies on a quantity we call the *maximum area*. Before defining and analyzing this quantity we first describe some notation.

Notation. The *arc* of a node N is the set of addresses in $\mathbf{A}$ between N's predecessor and itself. Note that the arc of a node depends on a configuration χ. More formally, we write $\mathsf{arc}_\chi(N) = (\mathsf{pred}_\chi(H_1(N)), \ldots, H_1(N)]$, where $\mathsf{pred}_\chi(N)$ is the *predecessor* function which assigns each address in $\mathbf{A}$ to its largest lower bound in χ. The *area* of a node N is defined as $\mathsf{area}(\chi, N) = |\mathsf{arc}_\chi(N)|$ and the area of a set of nodes $S \subseteq \chi$ is $\mathsf{area}(\chi, S) = \sum_{N \in S} \mathsf{area}(\chi, N)$. We denote by $\mathsf{maxareas}(\chi, x)$, the sum of the areas of x largest arcs in configuration χ. The *maximum area* of a configuration χ is equal to $\mathsf{maxareas}(\chi, \theta)$. As we will see, the maximum area is central not only to analyzing the balance of Chord but also to analyzing its stability.

Preliminaries. We now recall a Theorem from Byers, Considine and Mitzenmacher [4] that will help us upper bound Chord's maximum area.

Theorem 5.1 ([4])**.** *Let $\mathbf{C} \subseteq \mathbf{N}$ be a set of active nodes. If the following conditions hold (where all the probabilities are over the coins of* Overlay*):*

1. *for some constant δ_1,*
$$\Pr\left[\mathsf{maxareas}(\chi_{\mathbf{C}}, 1) \leq \frac{\delta_1 |\mathbf{A}| \log |\mathbf{C}|}{|\mathbf{C}|}\right] \leq 1 - p_1$$
2. *For suitable constants $\delta_2, \delta_3, \delta_4 > 0$, and $2 \leq c \leq \delta_4 \log |\mathbf{C}|$,*
$$\Pr\left[\left|\left\{\alpha \in A : |\alpha| \geq \frac{c|\mathbf{A}|}{|\mathbf{C}|}\right\}\right| \leq \frac{\delta_2 |\mathbf{C}|}{e^{c/\delta_3}}\right] \geq 1 - p_2$$
where A is the set of all arcs in $\chi_{\mathbf{C}}$.

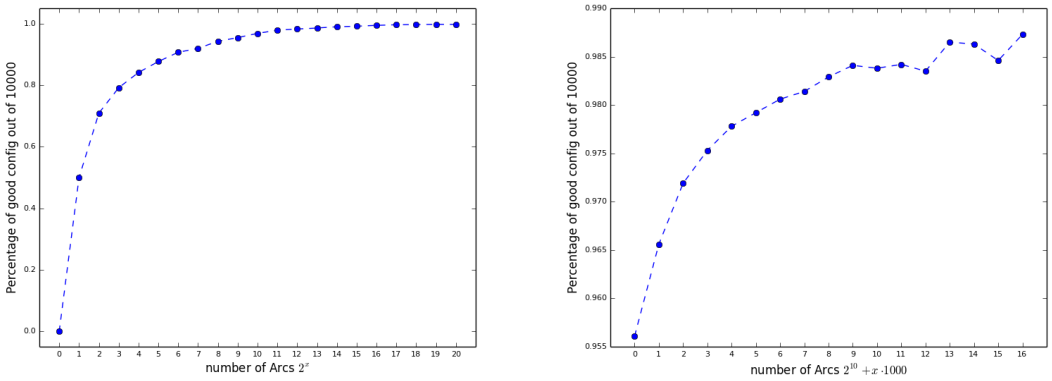
then, for all $\theta \leq c_2 |\mathbf{C}|$
$$\Pr\left[\mathsf{maxareas}(\chi_{\mathbf{C}}, \theta) \leq \frac{\gamma_1 |\mathbf{A}| \theta}{|\mathbf{C}|} \log \frac{|\mathbf{C}|}{\theta}\right] \geq 1 - p_1 - p_2 \cdot \log |\mathbf{C}|$$
where
$$\gamma_1 = 2\delta_3 + \frac{\delta_1}{1 - \frac{\delta_4}{2\delta_3}}, \quad \text{and} \quad c_2 = \min(2\delta_2 e^{-2/\delta_3}, 1/e).$$

To use Theorem 5.1 to bound Chord's maximum area, we need to find the constants for which Chord satisfies the Theorem's two conditions. We do this using the following Lemmas. The first is by Wang and Loguinov [30] and upper bounds the size of Chord's maximum arc (i.e. $\mathsf{maxareas}(\chi_{\mathbf{C}}, 1)$).

Lemma 5.2 ([30])**.** *Let $\mathbf{C} \subseteq \mathbf{N}$ be a set of active nodes. Then,*
$$\Pr\left[\mathsf{maxareas}(\chi_{\mathbf{C}}, 1) \leq \frac{(1 + c_1)|\mathbf{A}| \log |\mathbf{C}|}{|\mathbf{C}|}\right] \geq 1 - \frac{1}{|\mathbf{C}|^{c_1}},$$
where the probability is over the coins of Overlay *(i.e., the choice of H_1).*

For the second condition, we recall another Lemma from [4] based on the negative dependence of the size of Chord's arcs.

Lemma 5.3 ([4])**.** *Let $\mathbf{C} \subseteq \mathbf{N}$ be a set of active nodes. For $2 \leq c \leq n$,*
$$\Pr\left[\left|\left\{\alpha \in A : |\alpha| \geq \frac{c|\mathbf{A}|}{|\mathbf{C}|}\right\}\right| \geq \frac{2|\mathbf{C}|}{e^c}\right] \leq e^{-|\mathbf{C}|e^{-c/3}}$$
where the probability is over the coins of Overlay *(i.e., the choice of H_1).*



(a) (b)

Figure 3: Probability of sampling a good configuration. We write a value x on x-axis to mean 2^x in Figure 3a while $2^{10} + x \cdot 1000$ in Figure 3b.

Finding the constants. From Theorem 5.1 and Lemmas 5.2 and 5.3, we have the following Corollary which upper bounds Chord's maximum area.

Corollary 5.4. *Let* $\mathbf{C} \subseteq \mathbf{N}$ *be a set of active nodes. Then, for* $\theta \leq |\mathbf{C}|/e$

$$\Pr\left[\mathsf{maxareas}(\chi_{\mathbf{C}}, \theta) \leq \frac{6|\mathbf{A}|\theta}{|\mathbf{C}|}\log\frac{|\mathbf{C}|}{\theta}\right] \geq 1 - \frac{1}{|\mathbf{C}|^2} - (e^{-\sqrt{|\mathbf{C}|}} \cdot \log|\mathbf{C}|).$$

Proof. Setting $c_1 = 2$ in Lemma 5.2, we get $\delta_1 = 3$, and $p_1 = 1/|\mathbf{C}|^2$. Setting $c = \delta_4 \log|\mathbf{C}|$ in Lemma 5.3, we get $\delta_2 = 2$, $\delta_3 = 1$, and $p_2 \approx e^{-|\mathbf{C}|^{1-\delta_4}} = e^{-\sqrt{|\mathbf{C}|}}$ (setting $\delta_4 = 0.5$). Substituting the values of $\delta_1 = 3$, $\delta_2 = 2$, $\delta_3 = 1$, $\delta_4 = 0.5$, we get $\gamma_1 = 6$ and $\gamma_2 = e$.
Therefore, from Theorem 5.1, for all $\theta \leq |\mathbf{C}|/e$,

$$\Pr\left[\mathsf{maxareas}(\chi_{\mathbf{C}}, \theta) \leq \frac{6|\mathbf{A}|\theta}{|\mathbf{C}|}\log\frac{|\mathbf{C}|}{\theta}\right] \geq 1 - \frac{1}{|\mathbf{C}|^2} - (e^{-\sqrt{|\mathbf{C}|}} \cdot \log|\mathbf{C}|)$$

■

Experimental evaluation of maximum area. In the above Corollary, the error probability of $O(1/|\mathbf{C}|^2)$ stems from the fact that Lemma 5.2 only bounds $\mathsf{maxareas}(\chi_{\mathbf{C}}, 1)$ with probability $1 - O(1/|\mathbf{C}|^2)$.

We ran two experiments to empirically study the probability that $\mathsf{maxareas}(\chi_{\mathbf{C}}, 1)$ is bounded by $(1 + c_1)|\mathbf{A}| \log|\mathbf{C}|/|\mathbf{C}|$. In both experiments, we set $|\mathbf{A}| = 2^{24}$ and $c_1 = 0$ and vary $|\mathbf{C}|$. Then, for each value of $|\mathbf{C}|$, we sample 10000 configurations as follows: we sample $|\mathbf{C}|$ points uniformly at random from $\mathbf{A}$, sort them and compute the length of the maximum arc $\mathsf{maxareas}(\chi_{\mathbf{C}}, 1)$. We then count the number of configurations for which $\mathsf{maxareas}(\chi_{\mathbf{C}}, 1) \leq (1 + c_1)|\mathbf{A}| \log|\mathbf{C}|/|\mathbf{C}| = |\mathbf{A}| \log|\mathbf{C}|/|\mathbf{C}|$. We call such configurations "good" configurations. This gives us the probability of sampling a good configuration for fixed $|\mathbf{A}|$, c_1 and $|\mathbf{C}|$. Note that we chose $c_1 = 0$ because this is the worst value of c_1: any configuration with $\mathsf{maxareas}(\chi_{\mathbf{C}}, 1)$ less than $|\mathbf{A}| \log|\mathbf{C}|/|\mathbf{C}|$ will also have $\mathsf{maxareas}(\chi_{\mathbf{C}}, 1)$ less than $(1 + c_1)|\mathbf{A}| \log|\mathbf{C}|/|\mathbf{C}|$, where $c_1 \geq 1$.

Figure 3a shows the probability of sampling a good configuration as the number of nodes (or correspondingly arcs) are doubled from 1 to 2^{20}, and Figure 3b shows the probability when the

number of nodes are incremented by 1000 starting from 2^{10} until $\sim 2^{14}$. We see in both plots of Figure 3 that the probability of sampling a good configuration increases exponentially as a function of the number of active nodes $|\mathbf{C}|$. Moreover, the probability of sampling a good configuration is approximately 0.96 when $|\mathbf{C}| \approx 2^{10} = 1024$. Therefore, our experiments suggest that for $|\mathbf{C}| \geq 1024$, the Overlay algorithm samples a good configuration with exponentially high probability.

5.2 The Balance of Chord

We are now ready to analyze the balance of Chord.

Theorem 5.5. *Let $\mathbf{C} \subseteq \mathbf{N}$ be a set of active nodes. If $\mathsf{maxareas}(\chi_{\mathbf{C}}, \theta) \leq \lambda$, then $\chi_{\mathbf{C}}$ is (ε, θ)-balanced with*

$$\varepsilon = \frac{\lambda}{|\mathbf{A}|} + \frac{2\lambda|\mathbf{C}|\log|\mathbf{C}|}{|\mathbf{A}|^2}.$$

Proof. Let $n = |\mathbf{C}|$. For all $\ell \in \mathbf{L}$, we define the event $\mathcal{E}_1$ as server of ℓ being one of the nodes in S, and $\mathcal{E}_2$ as one of the nodes in S being on the path to server of ℓ. Precisely,

$$\mathcal{E}_1 = \{\mathsf{succ}_{\chi_{\mathbf{C}}}(H_2(\ell)) \in \cup_{N\in S} H_1(N)\}$$
$$\mathcal{E}_2 = \left\{ \left(\cup_{N\in S}\, H_1(N) \bigcap \mathsf{route}_{\chi_{\mathbf{C}}}\big(r, \mathsf{succ}_{\chi_{\mathbf{C}}}(H_2(\ell))\big) \right) \neq \emptyset \right\}$$

For Chord, we then have that,

$$\Pr\left[\, \mathsf{server}_{\chi_{\mathbf{C}}}(\ell) \in \mathsf{Vis}_{\chi_{\mathbf{C}}}(r, S) \,\right] = \Pr\left[\, \mathcal{E}_1 \vee \mathcal{E}_2 \,\right] = \Pr\left[\, \mathcal{E}_1 \,\right] + \Pr\left[\, \mathcal{E}_2 \,\right], \tag{4}$$

where the second equality is because the two events are mutually exclusive. Since $\mathsf{maxareas}(\chi_{\mathbf{C}}, \theta) \leq \lambda$, the sum of the arcs of any θ nodes is at most λ. Therefore,

$$\Pr\left[\, \mathcal{E}_1 \,\right] = \Pr\left[\, H_2(\ell) \in \cup_{N\in S} \mathsf{arc}_{\chi_{\mathbf{C}}}(N) \,\right] = \frac{|\cup_{N\in S}\, \mathsf{arc}_{\chi_{\mathbf{C}}}(N)|}{|\mathbf{A}|} \leq \frac{\lambda}{|\mathbf{A}|}. \tag{5}$$

We now turn to event $\mathcal{E}_2$. We have by the union bound and the law of total probability that,

$$\begin{aligned}
\Pr\left[\, \mathcal{E}_2 \,\right] &\leq \sum_{N\in S} \Pr\left[\, H_1(N) \in \mathsf{route}_{\chi_{\mathbf{C}}}\big(r, \mathsf{succ}_{\chi_{\mathbf{C}}}(H_2(\ell))\big) \,\right] \\
&= \sum_{N\in S} \sum_{N'\in\mathbf{C}} \Pr\left[\, \mathsf{succ}_{\chi_{\mathbf{C}}}(H_2(\ell)) = H_1(N') \,\right] \cdot \\
&\quad \Pr\left[H_1(N) \in \mathsf{route}_{\chi_{\mathbf{C}}}(r, H_1(N')) \,\middle|\, \mathsf{succ}_{\chi_{\mathbf{C}}}(H_2(\ell)) = H_1(N') \right] \\
&= \sum_{N\in S} \sum_{N'\in\mathbf{C}} \frac{|\mathsf{arc}_{\chi_{\mathbf{C}}}(N')|}{|\mathbf{A}|} \Pr\left[H_1(N) \in \mathsf{route}_{\chi_{\mathbf{C}}}(r, H_1(N')) \,\middle|\, \mathsf{succ}_{\chi_{\mathbf{C}}}(H_2(\ell)) = H_1(N') \right], \quad (6)
\end{aligned}$$

where the last equation follows from Eq. (5). Let $\mathcal{E}_3$ be the event that

$$\{H_1(N) \in \mathsf{route}_{\chi_{\mathbf{C}}}(r, H_1(N')) \mid \mathsf{succ}_{\chi_{\mathbf{C}}}(H_2(\ell)) = H_1(N')\}$$

and let $r_1, \ldots, r_m$ be the addresses in $\mathsf{route}_{\chi_{\mathbf{C}}}(r, H_1(N'))$. We then have,

$$\begin{aligned}
\Pr[\,\mathcal{E}_3\,] &= \Pr\left[\, H_1(N) \in \{r_1, \ldots, r_m\} \;\middle|\; \mathsf{succ}_{\chi_{\mathbf{C}}}(H_2(\ell)) = H_1(N') \,\right] \\
&\leq \sum_{i=1}^{m} \Pr\left[\, H_1(N) = r_i \;\middle|\; \mathsf{succ}_{\chi_{\mathbf{C}}}(H_2(\ell)) = H_1(N') \,\right] \\
&= \frac{m}{|\mathbf{A}|} \\
&\leq \frac{\log n}{|\mathbf{A}|},
\end{aligned}$$

where the last inequality follows from the fact that the path length in Chord can be at most $\log n$. Substituting this in Eq. (6) we get,

$$\begin{aligned}
\Pr[\,\mathcal{E}_2\,] &\leq \sum_{N \in S} \sum_{N' \in \mathbf{C}} \frac{|\mathsf{arc}_{\chi_{\mathbf{C}}}(N')|}{|\mathbf{A}|} \cdot \frac{\log n}{|\mathbf{A}|} \\
&= \frac{\theta \log n}{|\mathbf{A}|} \cdot \sum_{N' \in \mathbf{C}} \frac{|\mathsf{arc}_{\chi_{\mathbf{C}}}(N')|}{|\mathbf{A}|} \\
&\leq \frac{\theta \log n}{|\mathbf{A}|} \cdot \frac{2n\lambda}{a|\mathbf{A}|} \\
&= \frac{2n\lambda \log n}{|\mathbf{A}|^2}, \qquad (7)
\end{aligned}$$

where the second to last inequality follows from the fact that there can be at most $n/\theta + 1 \leq 2n/\theta$ sets of size θ within $\mathbf{C}$ and the sum of the arcs of nodes in each set is at most λ. Finally, the Theorem follows by plugging Eqs. (5) and (7) into Eq. (4).

■

Theorem 5.6. *Let $\mathbf{C}$ be a set of active nodes. For all $\theta \leq |\mathbf{C}|/e$, Chord is $(\varepsilon, \delta, \theta)$-balanced for*

$$\varepsilon = \frac{6\theta}{|\mathbf{C}|} \log\left(\frac{|\mathbf{C}|}{\theta}\right)\left(1 + \frac{2|\mathbf{C}| \log |\mathbf{C}|}{|\mathbf{A}|}\right), \quad \text{and} \quad \delta = \frac{1}{|\mathbf{C}|^2} + (e^{-\sqrt{|\mathbf{C}|}} \cdot \log |\mathbf{C}|),$$

Proof. From Corollary 5.4, we know that for $\theta \leq |\mathbf{C}|/e$,

$$\Pr[\,\mathsf{maxareas}(\chi_{\mathbf{C}}, \theta) \leq \lambda\,] \geq 1 - \delta \quad \text{for} \quad \lambda = \frac{6|\mathbf{A}|\theta}{|\mathbf{C}|} \log \frac{|\mathbf{C}|}{\theta}$$

and δ as stated above in theorem statement.
Therefore, from Lemma 5.5, we conclude that for $\theta \leq |\mathbf{C}|/e$,

$$\Pr[\,(H_1, \mathbf{C}) \text{ is } (\varepsilon, \theta)\text{-balanced}\,] \geq 1 - \delta \quad \text{for} \quad \varepsilon = \frac{\lambda}{|\mathbf{A}|} + \frac{2\lambda|\mathbf{C}| \log |\mathbf{C}|}{|\mathbf{A}|^2}$$

Substituting the value of λ in last equation, we conclude the proof.

■

Remark. It follows from Theorem 5.6 that if $2|\mathbf{C}|\log|\mathbf{C}| < |\mathbf{A}|$, then

$$\varepsilon = O\left(\frac{\theta}{|\mathbf{C}|}\log\left(\frac{|\mathbf{C}|}{\theta}\right)\right)$$

and $\delta = O(1/|\mathbf{C}|^2)$. Note that assigning labels uniformly at random to nodes would achieve $\varepsilon = \theta/|\mathbf{C}|$ so Chord balances data fairly well as long as $2|\mathbf{C}|\log|\mathbf{C}| < |\mathbf{A}|$. If we set the Chord address space to be the set of 256-bit strings, then $2|\mathbf{C}|\log|\mathbf{C}| < |\mathbf{A}|$ holds even with 2^{240} active nodes so, for all practical purposes, this condition always holds.

5.3 The Security of our Chord-based EDHT

In the following Corollary we formally state the security of the standard scheme when its underlying DHT is instantiated with Chord.

Corollary 5.7. *If* $|\mathbf{L}| = \Theta(2^k)$, $|I| \leq |\mathbf{C}|/e$, *and if* EDHT *is instantiated with Chord, then it is* $\mathcal{L}_\varepsilon$*-secure with probability at least* $1 - 1/|\mathbf{C}|^2 - (e^{-\sqrt{|\mathbf{C}|}} \cdot \log|\mathbf{C}|) - \mathsf{negl}(k)$ *in the random oracle model, where*

$$\varepsilon = \frac{6|I|}{|\mathbf{C}|}\log\left(\frac{|\mathbf{C}|}{|I|}\right)\left(1 + \frac{2|\mathbf{C}|\log|\mathbf{C}|}{|\mathbf{A}|}\right).$$

Proof. The corollary follows from Theorem 4.2, Corollary 5.6 and the fact that Chord has non-committing allocations when H_2 is modeled as a random oracle. Note that during the simulation, the probability that $\mathcal{A}$ queries H_2 on at least one of the strings $t_1, \ldots, t_q$ is at most $\mathsf{poly}(k)/|\mathbf{L}|$. This is because $\mathcal{A}$ is polynomially-bounded so it can make at most $\mathsf{poly}(k)$ queries to H_2. And since for all i, $t_i = f(\ell_i)$, where f is a random function, the probability that $\mathcal{A}$ queries H_2 on at least one of $t_1, \ldots, t_q$ is at most $\mathsf{poly}(k)/|\mathbf{L}|$. And since $|\mathbf{L}| = \Theta(2^k)$, this probability is negligible in k.

■

From the discussion of Theorem 5.6, we know that if $2|\mathbf{C}|\log|\mathbf{C}| < |\mathbf{A}|$, then

$$\varepsilon = O\left(\frac{|I|}{|\mathbf{C}|}\log\left(\frac{|\mathbf{C}|}{|I|}\right)\right)$$

and $\delta = O(1/|\mathbf{C}|^2)$. Setting $|I| = |\mathbf{C}|/\alpha$, for some $\alpha \geq e$, we have $\varepsilon = O(\log(\alpha)/\alpha)$. Recall that, on each query, the leakage function leaks the query equality with probability at most ε. So, intuitively, this means that if an α fraction of nodes are corrupted then, the adversary can expect to learn the query equality of an $O(\log(\alpha)/\alpha)$ fraction of client queries. Note that this confirms the intuition that distributing an STE scheme suppresses its leakage.

6 Encrypted Distributed Hash Tables in the Transient Setting

In this section we define the security of transient EDHTs and analyze the security of the standard construction in this setting.

Functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}$

$\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}$ stores a dictionary DX initialized to empty, a set $\mathbf{C} \subseteq \mathbf{N}$ of active nodes, and a set $I \subseteq \mathbf{N}$ of corrupted nodes. It proceeds as follows, running with client $\mathcal{C}$, n active nodes in $\mathbf{C}$ and a simulator Sim:

- $\mathsf{Put}(\ell, v)$: Upon receiving a label/value pair (ℓ, v) from client $\mathcal{C}$, it sets $\mathsf{DX}[\ell] := v$, and sends the leakage $\mathcal{L}(\mathsf{DX}, (\mathtt{put}, \ell, v))$ to the simulator Sim.
- $\mathsf{Get}(\ell)$: Upon receiving a label ℓ from client C, it returns $\mathsf{DX}[\ell]$ to the client $\mathcal{C}$ and the leakage $\mathcal{L}(\mathsf{DX}, (\mathtt{get}, \ell, \perp))$ to the simulator Sim.
- $\mathsf{Leave}(N)$: Upon receiving $N \in \mathbf{C}$, it returns the leakage $\mathcal{L}(\mathsf{DX}, (\mathtt{leave}, N))$ to the simulator Sim and updates its set $\mathbf{C}$.
- $\mathsf{Join}(N)$: Upon receiving $N \in \mathbf{N} \setminus \mathbf{C}$, it returns the leakage $\mathcal{L}(\mathsf{DX}, (\mathtt{join}, N))$ to the simulator Sim and updates its set $\mathbf{C}$.

Figure 4: $\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}$: The EDHT^+ functionality parameterized with leakage function $\mathcal{L}$.

6.1 Syntax and Security Definitions

Syntax. A transient EDHT is a collection of eight algorithms $\mathsf{EDHT}^+ = (\mathsf{Gen}, \mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get}, \mathsf{Leave}, \mathsf{Join})$. The first six algorithms are the same as the perpetual setting. The seventh is an algorithm Leave executed by an existing node in network when it wishes to leave the network, whereas the eighth is an algorithm Join executed by a node willing to join the network. Both of them take nothing as input and output nothing but either halt the Daemon algorithm or make changes to the routing tables.

We assume in this work that when a node leaves the network, all the pairs stored at that node are "re-put" in the network and when a node joins the network all the pairs currently in the network are "re-put". We note that this is not the most efficient way to handle leaves and joins but in this work our focus is on security rather than efficiency and this strategy has the worst possible leakage.

Security. We formalize the security definition using the Real/Ideal-world paradigm. As in the perpetual case, we parametrize the definition with a stateful leakage function $\mathcal{L}$ that captures the information leaked by the Put, Get, Leave and Join operations.

The real-world experiment. The experiment is executed between a trusted party $\mathcal{T}$, a client $\mathcal{C}$, the set of all nodes $\mathbf{N}$, an environment $\mathcal{Z}$ and an adversary $\mathcal{A}$. The trusted party runs $\omega \leftarrow \mathsf{Overlay}(|\mathbf{N}|)$, and $\psi \leftarrow \mathsf{Alloc}(|\mathbf{N}|, \omega)$ and sends (ω, ψ) to all parties, i.e., the nodes, the client, the environment and the adversary. Given $z \in \{0,1\}^*$, the environment $\mathcal{Z}$ sends to the adversary $\mathcal{A}$, a subset $I \subseteq \mathbf{N}$ of nodes to corrupt. The client $\mathcal{C}$ generates a key $K \leftarrow \mathsf{Gen}(1^k)$. $\mathcal{Z}$ then selects and activates a set of nodes $\mathbf{C} \subseteq \mathbf{N}$ and adaptively chooses a polynomial number of operations op_j,

- If $\mathsf{op}_j = (\mathtt{get}, \ell)$, it sends op_j to $\mathcal{C}$ who executes $\mathsf{EDHT}^+.\mathsf{Get}(K, \ell)$.
- If $\mathsf{op}_j = (\mathtt{put}, \ell, v)$, it sends op_j to $\mathcal{C}$ who executes $\mathsf{EDHT}^+.\mathsf{Put}(K, \ell, v)$.
- If $\mathsf{op}_j = (\mathtt{leave}, N)$ with $N \in \mathbf{C}$, it sends $\mathtt{leave}$ to N, updates $\mathbf{C} = \mathbf{C} \setminus \{N\}$. The node N then executes $\mathsf{EDHT}^+.\mathsf{Leave}()$.
- If $\mathsf{op}_j = (\mathtt{join}, N)$ with $N \in \mathbf{N} \setminus \mathbf{C}$, it sends $\mathtt{join}$ to N and updates $\mathbf{C} = \mathbf{C} \cup \{N\}$. The node N then executes $\mathsf{EDHT}^+.\mathsf{Join}()$.

The client and all the nodes forward their outputs to $\mathcal{Z}$. $\mathcal{A}$ computes an arbitrary message m from its view and sends it to $\mathcal{Z}$. Finally, $\mathcal{Z}$ returns a bit that is output by the experiment. We denote by $\mathbf{Real}_{\mathcal{A},\mathcal{Z}}(k)$ the random variable that denotes $\mathcal{Z}$'s output bit.

The ideal-world experiment. The experiment is executed between a trusted party $\mathcal{T}$, a client $\mathcal{C}$, a set of nodes $\mathbf{N}$, an environment $\mathcal{Z}$ and a simulator Sim. Each party also has access to the ideal functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}$. Given $z \in \{0,1\}^*$, the environment $\mathcal{Z}$ selects a subset $I \subseteq \mathbf{N}$ of nodes to corrupt. $\mathcal{Z}$ then selects and activates a set of nodes $\mathbf{C} \subseteq \mathbf{N}$ and sends $(\mathbf{C}, I)$ to Sim and $\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}$. $\mathcal{Z}$ then adaptively chooses a polynomial number of operations op_j,

- If $\mathsf{op}_j = (\mathtt{get}, \ell)$, it sends op_j to $\mathcal{C}$ who forwards it to $\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}$. The functionality executes $\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}.\mathsf{Get}(\ell)$.
- If $\mathsf{op}_j = (\mathtt{put}, \ell, v)$, it sends op_j to $\mathcal{C}$ who forwards it to $\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}$. The functionality executes $\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}.\mathsf{Put}(\ell, v)$.
- If $\mathsf{op}_j = (\mathtt{leave}, N)$ with $N \in \mathbf{C}$, it updates $\mathbf{C} = \mathbf{C} \setminus \{N\}$ and sends $(\mathtt{leave}, N)$ to $\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}$. The functionality executes $\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}.\mathsf{Leave}(N)$.
- If $\mathsf{op}_j = (\mathtt{join}, N)$ with $N \in \mathbf{N} \setminus \mathbf{C}$, it updates $\mathbf{C} = \mathbf{C} \cup \{N\}$ and sends $(\mathtt{join}, N)$ to $\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}$. The functionality executes $\mathcal{F}^{\mathcal{L}}_{\mathsf{EDHT}^+}.\mathsf{Join}(N)$.

The client $\mathcal{C}$ and all the nodes forward their outputs to $\mathcal{Z}$. Sim sends an arbitrary message to $\mathcal{Z}$. Finally, $\mathcal{Z}$ returns a bit that is output by the experiment. We denote by $\mathbf{Ideal}_{\mathsf{Sim},\mathcal{Z}}(k)$ the random variable that denotes $\mathcal{Z}$'s output bit.

Definition 6.1 ($\mathcal{L}$-security)**.** *We say that a transient encrypted distributed hash table* $\mathsf{EDHT}^+ = (\mathsf{Gen}, \mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get}, \mathsf{Leave}, \mathsf{Join})$ *is $\mathcal{L}$-secure, if for all* PPT *adversaries $\mathcal{A}$ and all* PPT *environments $\mathcal{Z}$, there exists a* PPT *simulator* Sim *such that for all $z \in \{0,1\}^*$,*

$$|\Pr[\mathbf{Real}_{\mathcal{A},\mathcal{Z}}(k) = 1] - \Pr[\mathbf{Ideal}_{\mathsf{Sim},\mathcal{F},\mathcal{Z}}(k) = 1]| \leq \mathsf{negl}(k)$$

6.2 The Standard EDHT in the Transient Setting

In the transient setting, the standard scheme is composed of eight algorithms $\mathsf{EDHT}^+ = (\mathsf{Gen}, \mathsf{Overlay}, \mathsf{Alloc}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get}, \mathsf{Leave}, \mathsf{Join})$. The first six algorithms are exactly the same as in EDHT. The Leave algorithm simply calls $\mathsf{DHT}^+.\mathsf{Leave}$ while the Join algorithm calls $\mathsf{DHT}^+.\mathsf{Join}$. We now turn to describing the leakage of this scheme. We start with a description of the leakage for join and leave operations and then discuss the leakage for put and get operations.

Join and leave leakage. Roughly speaking, during the execution of the scheme, the adversary sees leakage on label/value pairs that are either stored at corrupted nodes or routed through corrupted nodes. In particular, this means that it does not receive any leakage about label/value pairs that are stored and routed through exclusively honest nodes. Now, when a join or leave operation occurs, label/value pairs are moved throughout the network (e.g., during a leave, the leaving node's pairs are redistributed to other nodes). At this point, the adversary could get new leakage about pairs that it had not seen before the leave/join operation. For example, this would occur if a previously unseen label/value pair (i.e., that was stored on the leaving node) gets routed through a corrupted node during the re-distribution.

To simulate a leave/join operation correctly, the simulator will have to correctly simulate the re-distribution of pairs including of pairs it has not seen yet. But at this stage, it does not even know how many such pairs exist. This is because it does not get executed on put operations for labels not stored or routed by corrupted nodes. To overcome this, we reveal to the simulator how many of these pairs exist through the leakage function.

This, however, affects the get and put leakages for these pairs: now that the pairs have been re-distributed to (or routed through) a corrupted node the adversary will receive get and put leakages on these pairs. There is a technical challenge here, which is that we do not know how to simulate *only* the pairs that are re-distributed to (or routed through) corrupted nodes, so to address this we additionally reveal to the simulator the leakage of all the previously unseen pairs. It is not clear if this is strictly necessary and it could be that the scheme achieves a "tighter" leakage function. Note that this does not affect new pairs, i.e., pairs that are added after the leave/join operation (until another leave/join operation occurs).

Note that by revealing the number κ of previously unseen pairs, one can compute the total number of put operations up to the last leave/join operation. We denote this value by τ and make it explicit in the leakage function for ease of exposition.

The leakage profile. We are now ready to formally describe the leakage profile achieved by the standard scheme in the transient setting.

- $\mathcal{L}_{\varepsilon}\Big(\mathsf{DX}, \Big\{(\mathsf{op}, \ell, v), (\mathsf{op}, N)\Big\}\Big)$:
  1. if $\mathsf{op} = \mathtt{get} \vee \mathtt{put}$ and ℓ has never been seen
     (a) sample and store $b_\ell \leftarrow \mathsf{Ber}(\varepsilon)$
  2. if $b_\ell = 1$
     (a) if $\mathsf{op} = \mathtt{put}$ output $(\mathtt{put}, \mathsf{qeq}(\ell))$
     (b) else if $\mathsf{op} = \mathtt{get}$ output $(\mathtt{get}, \mathsf{qeq}(\ell))$
  3. else if $b_\ell = 0$
     (a) Increment κ if $\mathsf{op} = \mathtt{put}$ and ℓ has never been seen
     (b) output $\perp$
  4. Increment τ
  5. if $\mathsf{op} = \mathtt{leave} \vee \mathtt{join}$
     (a) output $(\mathsf{op}, N, \kappa, \tau)$
     (b) set $b_\ell = 1$ for all the put labels that have been seen in the past
     (c) reset κ to 0

We now show that EDHT^+ is $\mathcal{L}_{\varepsilon}$-secure in the transient setting with probability negligibly close to $1 - \delta$ when its underlying transient DHT is $(\varepsilon, \delta, \theta)$-balanced and is non-committing.

Theorem 6.2. *If $|I| \leq \theta$ and DHT^+ is $(\varepsilon, \delta, \theta)$-balanced and has non-committing and label-independent allocations, then EDHT^+ is $\mathcal{L}_{\varepsilon}$-secure with probability at least $1 - \delta - \mathsf{negl}(k)$.*

Proof. Consider the simulator Sim that works as follows. Given a set of corrupted nodes $I \subseteq \mathbf{N}$, and a set of active nodes $\mathbf{C} \subseteq \mathbf{N}$, it first computes $\omega \leftarrow \mathsf{DHT}^+.\mathsf{Overlay}(n)$, initializes n nodes $N_1, \ldots, N_n$ in $\mathbf{C}$, simulates the adversary $\mathcal{A}$ with I and $\mathbf{C}$ as input, and generates a symmetric key

$K \leftarrow \mathsf{SKE.Gen}(1^k)$. It then sets $I' = \mathbf{C} \cap I$, $B = \mathsf{Vis}_{\omega,\mathbf{C}}(s, I')$, $G = \mathbf{A} \setminus B$, and $p' = \Pr[\,\psi(\ell) \in B\,]$. If $p' > \varepsilon$, the simulator aborts, otherwise it continues. The simulator also initializes two empty multimaps MM and MM'.

When a leave/join operation is executed, the simulator receives from $\mathcal{F}_{\mathsf{EDHT}^+}$ the leakage

$$\lambda \in \left\{ \Big(\mathtt{leave}, N, \kappa, \tau \Big), \Big(\mathtt{join}, N, \kappa, \tau \Big) \right\}.$$

For each $j \in [\kappa]$, it sets $t_j \xleftarrow{\$} \{0,1\}^d$ and $e_j \leftarrow \mathsf{SKE.Enc}(K, 0)$, samples an address $a \leftarrow \Delta_{\omega,\mathbf{C}}(G)$, programs ψ to map t to a, computes $N' \leftarrow \mathsf{server}(t_j)$, and adds (t_j, e_j) to $\mathsf{MM}[N']$. It then sets $\mathsf{MM}'[\tau' \| \tau] = \{t_1, \ldots, t_\kappa\}$, where τ' is the time of the last leave/join operation. It also sets $b'_i = 1$ for all the put labels that have been seen in the past. Finally, if the operation is a leave operation, it updates $\mathbf{C} = \mathbf{C} \setminus \{N\}$, updates the routing tables to exclude N, and executes $\mathsf{DHT.Put}(t, v)$ on all the (t, v) pairs stored in $\mathsf{MM}[N]$, updating MM according to how pairs move.

If the operation is a join operation, it updates $\mathbf{C} = \mathbf{C} \cup \{N\}$, updates the routing tables to include N, and executes $\mathsf{DHT.Put}(t, v)$ on all the (t, v) pairs stored in MM for all the nodes, updating MM according to how pairs move. It finally, resets $\mathsf{MM}[N]$ to $\perp$, $I' = I \cap \mathbf{C}$, $B = \mathsf{Vis}_{\omega,\mathbf{C}}(s, I')$, $G = \mathbf{A} \setminus B$, and computes $p' = \Pr[\,\psi(\ell) \in B\,]$. If $p' > \varepsilon$, it aborts and exits, otherwise it continues.

When a put/get operation is executed, the simulator receives from $\mathcal{F}_{\mathsf{EDHT}^+}$ leakage

$$\lambda \in \left\{ \Big(\mathtt{put}, \mathsf{qeq}(\ell) \Big), \Big(\mathtt{get}, \mathsf{qeq}(\ell) \Big), \perp \right\}.$$

If $\lambda = \perp$ then Sim does nothing. If $\lambda \neq \perp$, then Sim checks the query equality to see if the label has been used in the past. If not, it samples and stores a bit

$$b' \leftarrow \mathsf{Ber}\left(\frac{p'}{\varepsilon}\right).$$

Note that, this is indeed a valid Bernoulli distribution since

$$p' = \Pr[\,\psi(\ell) \in B\,] = \Pr\left[\,\mathsf{server}_{\omega,\psi}(\ell) \in \mathsf{Vis}_{\omega,\mathbf{C}}(s, I')\,\right] \leq \Pr\left[\,\mathsf{server}_{\omega,\psi}(\ell) \in \mathsf{Vis}_{\omega,\mathbf{C}}(s, I)\,\right] \leq \varepsilon,$$

where the second equality follows from the definition of visible address, and the last two inequalities follows from $|I'| \leq |I| \leq \theta$ and $(\omega, \mathbf{C})$ being (ε, θ)-balanced.

It then sets $t \xleftarrow{\$} \{0,1\}^d$ and computes $e \leftarrow \mathsf{SKE.Enc}(K, 0)$. If $b' = 0$, and the operation is a put operation, it samples $a \leftarrow \Delta_{\omega,\mathbf{C}}(G)$, otherwise, (if $b' = 1$ and irrespective of operation) it samples $a \leftarrow \Delta_{\omega,\mathbf{C}}(B)$. In either case, it programs ψ to map t to a, computes $N' \leftarrow \mathsf{server}(t_j)$, adds (t_j, e_j) to $\mathsf{MM}[N']$, and executes $\mathsf{DHT.Put}(t, e)/\mathsf{DHT.Get}(t)$ depending on whether the operation was a put or a get.

If, on the other hand, the label has been used in the past (as deduced from query equality), it retrieves the bit b' previously sampled. If $b' = 0$, it does nothing. If $b' = 1$, it sets t to the d-bit value previously used and $e \leftarrow \mathsf{SKE.Enc}(K, 0)$, and executes $\mathsf{DHT.Put}(t, e)/\mathsf{DHT.Get}(t)$. If $b' = \perp$, (this occurs for the labels which b was 0 initially but later leave/join occured), it sets $t = t'$, where $t' \xleftarrow{\$} \mathsf{MM}'[\tau \| \tau']$, such that $\mathsf{qeq}(\ell) \in [\tau, \tau']$, and then removes t' from $\mathsf{MM}'[\tau \| \tau']$. It finally computes $e \leftarrow \mathsf{SKE.Enc}(K, 0)$ and executes $\mathsf{DHT.Put}(t, e)/\mathsf{DHT.Get}(t)$.

Once all of the environment's operations are processed, the simulator returns whatever the adversary outputs.

It remains to show that the view of the adversary $\mathcal{A}$ during the simulation is indistinguishable from its view in a **Real** experiment. We do this using a sequence of games.

Game_0 : is the same as a $\mathbf{Real}_{\mathcal{A},\mathcal{Z}}(k)$ experiment.

Game_1 : is the same as Game_0 except that the encryption of the value v during a Put is replaced by $\mathsf{SKE.Enc}(K_2, 0)$.

Game_2 : is the same as Game_1 except that output of the PRF F is replaced by a truly random string of d bits.

Game_3 : is the same as Game_2 except that for each operation op, if $\mathsf{op} \in \{(\mathtt{get}, \ell), (\mathtt{put}, \ell, v)\}$, we check if the label ℓ has been seen before. If not, we sample and store a bit $b_\ell \leftarrow \mathsf{Ber}(\varepsilon)$, else we set b_ℓ to the bit previously sampled for ℓ. If $b_\ell = 1$ and $\mathsf{op} = (\mathtt{put}, \ell, v)$, we replace the Put operation with $\mathsf{Sim}(\mathtt{put}, \mathsf{qeq}(\ell))$ and if $\mathsf{op} = (\mathtt{get}, \ell)$ we replace the Get operation with $\mathsf{Sim}(\mathtt{get}, \mathsf{qeq}(\ell))$. If however $\mathsf{op} = (\mathtt{leave}, N)$, we replace the Leave operation with $\mathsf{Sim}(\mathtt{leave}, N, \kappa, \tau)$ and set $b_\ell = 1$ for all the put labels that have been seen in the past. Similarly if $\mathsf{op} = (\mathtt{join}, N)$, we replace the Join operation with $\mathsf{Sim}(\mathtt{join}, N, \kappa, \tau)$ and set $b_\ell = 1$ for all the put labels that have been seen in the past.

Game_1 is indistinguishable from Game_0, otherwise the encryption scheme is not semantically secure. Game_2 is indistinguishable from Game_1 because the outputs of pseudorandom functions are indistinguishable from random strings.

Let $(\omega, \mathbf{C})$ be the current overlay. Since DHT is $(\varepsilon, \delta, \theta)$-balanced, with probability at least $1 - \delta$, for all $\mathbf{C} \subseteq \mathbf{N}$, $(\omega, \mathbf{C})$ will be (ε, θ)-balanced. Furthermore, as shown in Theorem 4.2, if it is (ε, θ)-balanced then $p' \leq p$. It follows then that the simulator aborts with probability at most δ so for the rest of the proof, we argue indistinguishability assuming (ε, θ)-balanced overlays.

As in the proof of Theorem 4.2, we will consider the views of nodes in I' for each operation and show them to be indistinguishable across Game_2 and Game_3. We will denote this by $\mathbf{view_2}^i(I')$ and $\mathbf{view_3}^i(I')$ for Game_2 and Game_3 respectively.

Let $\mathbf{op}$ denote the sequence of operations generated by the environment. To prove the indistinguishability of views, we divide the operations in $\mathbf{op}$ into buckets where the bucket boundaries are the leave/join operations.

Now consider the first bucket. Since no leaves/joins have yet been simulated, b'_i can only be 0 or 1 but not $\perp$. Notice that for get and put operations in the bucket, when $b'_i = 1$, the simulator programs ψ in the same way as the simulator of Theorem 4.2. It does some extra bookkeeping in addition but that does not affect the view of the nodes in set I' for that operation. Moreover, for put operations when $b'_i = 0$, it only programs ψ to addresses not visible to I' and does nothing else which generates any extra view for nodes in I'. Therefore, using the same argument as in Theorem 4.2, we conclude that for get and put operations the views are indistinguishable.

Let op_i be the first leave/join operation (boundary of the first bucket) and let $t_1, \ldots, t_q$ be the distinct labels of put operations in first bucket. Now let A_r be the random variable denoting the

allocation of $t_1, \ldots, t_q$ to addresses in $\mathbf{view_2}$. Then, using the law of total probability, we get

$$\begin{aligned}
&\Pr\left[\, \mathbf{view_2}^i(I') = \mathbf{v} \,\right] \\
&= \sum_{(\alpha_1,\ldots,\alpha_q)\in\mathbf{A}^q} \Pr\left[\, \mathbf{view_2}^i(I') = \mathbf{v} \mid A_r = (\alpha_1,\ldots,\alpha_q) \,\right] \cdot \Pr\left[\, A_r = (\alpha_1,\ldots,\alpha_q) \,\right] \qquad (8)
\end{aligned}$$

Similarly, let A_s be the random variable denoting the allocation of $t_1, \ldots, t_q$ to addresses in $\mathbf{view_3}$. Then,

$$\begin{aligned}
&\Pr\left[\, \mathbf{view_3}^i(I') = \mathbf{v} \,\right] \\
&= \sum_{(\alpha_1,\ldots,\alpha_q)\in\mathbf{A}^q} \Pr\left[\, \mathbf{view_3}^i(I') = \mathbf{v} \mid A_s = (\alpha_1,\ldots,\alpha_q) \,\right] \cdot \Pr\left[\, A_s = (\alpha_1,\ldots,\alpha_q) \,\right]
\end{aligned}$$

But conditioned on a fixed allocation $(\alpha_1, \ldots, \alpha_q) \in \mathbf{A}^q$ of labels, during leave/join operations, the views of the nodes in I' will be the same in both the games, since both of them will be re-distributing the same number of pairs using $\mathsf{DHT.Put}$. Therefore,

$$\Pr\left[\, \mathbf{view_2}^i(I') = \mathbf{v} \mid A_r = (\alpha_1,\ldots,\alpha_q) \,\right] = \Pr\left[\, \mathbf{view_3}^i(I') = \mathbf{v} \mid A_s = (\alpha_1,\ldots,\alpha_q) \,\right] \qquad (9)$$

Next we show that,

$$\Pr\left[\, A_r = (\alpha_1,\ldots,\alpha_q) \,\right] = \Pr\left[\, A_s = (\alpha_1,\ldots,\alpha_q) \,\right]$$

Notice that we can rewrite [3]

$$\Pr\left[\, A_r = (\alpha_1,\ldots,\alpha_q) \,\right] = \prod_{j\in[q]} \Pr\left[\, \psi(t_j) = \alpha_j \,\right] = \prod_{j\in[q]} \Pr\left[\, \psi(\ell) = \alpha_j \,\right]$$

where the last equality follows because ψ is a label-independent allocation function. The allocation in Game_3 is determined by the programmed ψ function. To avoid any confusion with the ψ function of Game_2, we denote by ψ_P, the programmed allocation function of Game_3. Then, we can rewrite,

$$\Pr\left[\, A_s = (\alpha_1,\ldots,\alpha_q) \,\right] = \prod_{j\in[q]} \Pr\left[\, \psi_P(t_j) = \alpha_j \,\right]$$

There are two subcases to consider. In the first case, $\alpha_j \in B$. Then,

$$\Pr\left[\, \psi_P(t_j) = \alpha_j \,\right] = \Pr\left[\, b_j = 1 \wedge b'_j = 1 \wedge a_j = \alpha_j \,\right]$$

where $a_j \leftarrow \Delta_{\omega,\mathbf{C}}(B)$. Now,

$$\begin{aligned}
\Pr\left[\, b_j = 1 \wedge b'_j = 1 \wedge a_j = \alpha_j \,\right] &= \varepsilon \cdot \frac{\Pr\left[\, \psi(\ell) \in B \,\right]}{\varepsilon} \cdot \frac{\Pr\left[\, \psi(\ell) = \alpha_j \,\right]}{\Pr\left[\, \psi(\ell) \in B \,\right]} \\
&= \Pr\left[\, \psi(\ell) = \alpha_j \,\right]
\end{aligned}$$

In the second case, $\alpha_j \in \mathbf{A} \setminus B = G$. Then,

$$\Pr\left[\, \psi_P(t_j) = \alpha_j \,\right] = \Pr\left[\, \mathcal{E}_1 \,\right] + \Pr\left[\, \mathcal{E}_2 \,\right]$$

[3] there is an implicit assumption made here that for each label, its allocation to an address is independent of the previous allocations. However, the proof can be extended when no such assumption is made using the chain rule of probability.

where

$$\Pr[\mathcal{E}_1] = \Pr\Big[b_j = 1 \wedge b'_j = 0 \wedge a_j = \alpha_j\Big], \text{ and}$$
$$\Pr[\mathcal{E}_2] = \Pr[b_j = 0 \wedge a_j = \alpha_j]$$

such that $a_j \leftarrow \Delta_{\omega,\mathbf{C}}(G)$. Then,

$$\begin{aligned}\Pr[\mathcal{E}_1] &= \Pr\Big[b_j = 1 \wedge b'_j = 0 \wedge a_j = \alpha_j\Big] \\ &= \varepsilon \cdot \left(1 - \frac{\Pr[\psi(\ell) \in B]}{\varepsilon}\right) \cdot \frac{\Pr[\psi(\ell) = \alpha_j]}{\Pr[\psi(\ell) \in G]}, \text{ and}\end{aligned}$$

$$\begin{aligned}\Pr[\mathcal{E}_2] &= \Pr[b_j = 0 \wedge a_j = \alpha_j] \\ &= (1-\varepsilon) \cdot \frac{\Pr[\psi(\ell) = \alpha_j]}{\Pr[\psi(\ell) \in G]}\end{aligned}$$

Adding the two probabilites, we get,

$$\begin{aligned}\Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2] &= \frac{\Pr[\psi(\ell) = \alpha_j]}{\Pr[\psi(\ell) \in G]} \cdot \left(\varepsilon \cdot \left(1 - \frac{\Pr[\psi(\ell) \in B]}{\varepsilon}\right) + \Big(1 - \varepsilon\Big)\right) \\ &= \frac{\Pr[\psi(\ell) = \alpha_j]}{\Pr[\psi(\ell) \in G]} \cdot \Big(1 - \Pr[\psi(\ell) \in B]\Big) \\ &= \frac{\Pr[\psi(\ell) = \alpha_j]}{\Pr[\psi(\ell) \in G]} \cdot \Pr[\psi(\ell) \in G] \\ &= \Pr[\psi(\ell) = \alpha_j]\end{aligned}$$

Hence,

$$\Pr[A_r = (\alpha_1, \ldots, \alpha_q)] = \Pr[A_s = (\alpha_1, \ldots, \alpha_q)] \tag{10}$$

Therefore, by substituting Equations 9 and 10 in Equation 8, we conclude that at the first churn operation,

$$\Pr\Big[\mathbf{view_2}^i(I') = \mathbf{v}\Big] = \Pr\Big[\mathbf{view_3}^i(I') = \mathbf{v}\Big]$$

Moreover, since the allocation distribution before the churn operation is the same and both the games use the same DHT.Put to move the pairs, therefore, the new allocation distribution will also be the same. Hence using induction on each bucket, we prove that views will be indistinguishable for all the buckets. The proof follows by noticing that Game_3 is same as $\mathbf{Ideal}_{\mathsf{Sim},\mathcal{Z}}(k)$ experiment.

■

Efficiency. The time, round and communication complexities of leave and join operations of the standard scheme is transient setting are the same as the underlying DHT.

7 A Chord-Based EDHT in the Transient Setting

We now describe and analyze how Chord can work in a transient setting. The Chord paper does not precisely specify how joins and leaves should be handled. More precisely, it describes what should happen to the pairs that are stored but not exactly how those pairs should get to their destination. Because of this, we describe here a simple approach based on "re-hashing". We note that this is not the most efficient way to handle leaves and joins but it is correct and our focus is on security rather than efficiency.

Leaves and joins in Chord. When a new node $N \in \mathbf{N} \setminus \mathbf{C}$ joins the network, it is first assigned an address $H_1(N) \in \mathbf{A}$. Then, the routing tables of all the other nodes are updated. Finally, all the label/value pairs stored at $\mathsf{succ}_{\chi_\mathbf{C}}(H_1(N))$ are re-hashed and stored at their new destination (if necessary). When a node $N \in \mathbf{C}$ leaves, the routing tables of all the other nodes are updated and all the label/value pairs stored at N are moved to the $\mathsf{succ}_{\chi_\mathbf{C}}(H_1(N))$.

7.1 Analysis of Chord's Stability

Recall from the security analysis of the Chord-based EDHT that its leakage was $\mathcal{L}_\varepsilon$, where ε is a function of the upper bound on $\mathsf{maxareas}$ and where the simulation error δ is a function of the probability of that bound.

In perpetual setting there is a single configuration corresponding to fixed set of active nodes. However, in transient setting there are multiple configurations – every time a node leaves/joins, the configuration changes. Therefore, in transient setting, the parameters ε and δ are functions of bounds on $\mathsf{maxareas}$ and their probabilities of each possible configuration.

We describe here, at a high level, two strategies for computing these parameters, with tradeoffs between quality of simulation and running time efficiency. The first strategy is efficient but has a δ which is $1/\mathsf{poly}(|\mathbf{N}|)$ while the second has expensive setup but improves δ to $\mathsf{negl}(k)$.

Approach #1. In this approach we upper bound the $\mathsf{maxareas}$ in the configuration $\chi_\mathbf{C}$ via the $\mathsf{maxareas}$ of the configuration $\chi_\mathbf{N}$. The approach relies on two main observations. The first is that any configuration $\chi_\mathbf{C}$ can be expressed as $\chi_\mathbf{N} \setminus \chi_{\mathbf{N}\setminus\mathbf{C}}$ which, intuitively, means we can recover $\chi_\mathbf{C}$ by starting with $\chi_\mathbf{N}$ (which includes every node in the name space) and removing the nodes $\mathbf{N} \setminus \mathbf{C}$. The second observation is that if we start with a given configuration $\chi_\mathbf{C}$ and remove a node N, then N's area becomes visible to some other (currently active) node.

But how exactly can we use these observations to bound the maximum area in $\chi_\mathbf{C}$ using the maximum area in $\chi_\mathbf{N}$? We start with $\chi_\mathbf{N}$ and remove the nodes in $\mathbf{N} \setminus \mathbf{C}$; but for each node N that is removed, we assume the worst-case and assign N's area to one of the θ nodes with largest arcs. The resulting area will be an upper bound on the true maximum area. More formally, we have that $\mathsf{maxareas}(\chi_\mathbf{C}, \theta) \leq \mathsf{maxareas}(\chi_\mathbf{N}, \theta + |\mathbf{N}| - |\mathbf{C}|)$.

For our purposes, we will need to show that this bound holds for all large enough $\mathbf{C}$'s so the next step will be to prove that if $|\mathbf{N}| - |\mathbf{C}| \leq d$, then for all $\mathbf{C}$ such that $|\mathbf{C}| \geq |\mathbf{N}| - d$, $\mathsf{maxareas}(\chi_\mathbf{N}, \theta + |\mathbf{N}| - |\mathbf{C}|)$ is upper bounded by $\mathsf{maxareas}(\chi_\mathbf{N}, \theta + d)$. But we can bound the latter using Corollary 5.4 with probability at least $1 - O(1/|\mathbf{N}|^2)$.

Approach #2. The limitation of the previous approach is that the bound only holds with probability $1 - O(1/|\mathbf{N}|^2)$ which leads to a $O(1/|\mathbf{N}|^2)$ error probability for the simulation. Using our second approach, however, we will reduce the error probability to be negligible.

We do this by using a new overlay algorithm $\overline{\mathsf{Overlay}}$ that works as follows. It runs the old $\mathsf{Overlay}$ algorithm $r = O(k/\log|\mathbf{N}|)$ times in the hope of sampling an overlay parameter $\omega = H_1$ such that $\mathsf{maxareas}(\chi_\mathbf{N}, 1)$ is small. We show in Lemma 7.4 that $\overline{\mathsf{Overlay}}$ will find such an H_1 with overwhelming probability in k.

Using $\overline{\mathsf{Overlay}}$, one can find, with overwhelming probability, an overlay with a small $\mathsf{maxareas}(\chi_\mathbf{N}, 1)$. This, in turn, gives us a bound on $\mathsf{maxareas}(\chi_\mathbf{N}, \theta + d)$ with overwhelming probability (Corollary 5.4) which yields a simulation with negligible error probability. As we will see, the main limitation of this approach is that $\overline{\mathsf{Overlay}}$ runs in time $O(k|\mathbf{N}|)$ as opposed to $\mathsf{Overlay}$ which runs in $O(1)$ time. We however show experimentally in Section 5.1, that the probability of sampling a good hash

function in a *single* trial is very high (and seems to grow exponentially). Therefore, for practical purposes, it is most likely enough to use $\mathsf{Overlay}$ instead of $\overline{\mathsf{Overlay}}$.

7.1.1 Approach #1: High Probability Simulation Success

Here, we analyze our first strategy. We start by proving a Lemma that bounds the maximum areas of all the configurations $\chi_{\mathbf{C}}$ with large enough $\mathbf{C}$.

Lemma 7.1. *If* $\chi_{\mathbf{C}} = (H_1, \mathbf{N})$ *is a configuration such that* $\Pr[\,\mathsf{maxareas}(\chi_{\mathbf{N}}, \theta + d) \leq \alpha\,] \geq 1 - \beta$ *then*

$$\Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq|\mathbf{N}|-d} \mathsf{maxareas}(\chi_{\mathbf{C}}, \theta) \leq \alpha\right] \geq 1 - \beta.$$

Proof. We prove this by contradiction. Suppose that there exists a set of active nodes $\mathbf{C}^*$ and a subset of nodes $S \subseteq \mathbf{C}^*$ such that $|\mathbf{C}^*| \geq |\mathbf{N}| - d$ and that $|S| = \theta$ for which $\mathsf{area}(\chi_{\mathbf{C}^*}, S) > \alpha$. We then show that there exists a set of nodes $D \subseteq \mathbf{N}$ of size $\theta + d$ such that $\mathsf{area}(\chi_{\mathbf{N}}, D) > \alpha$.

Consider the set $D = S \cup \mathbf{N} \setminus \mathbf{C}^*$ and note that

$$\mathsf{area}(\chi_{\mathbf{N}}, D) = \mathsf{area}(\chi_{\mathbf{N}}, S \cup \mathbf{N} \setminus \mathbf{C}^*) = \mathsf{area}(\chi_{\mathbf{N}}, S) + \mathsf{area}(\chi_{\mathbf{N}}, \mathbf{N} \setminus \mathbf{C}^*) \tag{11}$$

We know that for some subset $Z \subseteq \mathbf{N} \setminus \mathbf{C}^*$, the following holds:

$$\begin{aligned}\mathsf{area}(\chi_{\mathbf{C}^*}, S) &= \mathsf{area}(\chi_{\mathbf{N}}, S) + \mathsf{area}(\chi_{\mathbf{N}}, Z) \\ &\leq \mathsf{area}(\chi_{\mathbf{N}}, S) + \mathsf{area}(\chi_{\mathbf{N}}, \mathbf{N} \setminus \mathbf{C}^*),\end{aligned} \tag{12}$$

where the equality holds because when nodes in $\mathbf{N} \setminus \mathbf{C}^*$ are removed from $\chi_{\mathbf{N}}$, their areas might become visible to nodes in S, and the inequality holds because $Z \subseteq \mathbf{N} \setminus \mathbf{C}^*$. From Equations 11 and 12, we conclude that

$$\mathsf{area}(\chi_{\mathbf{N}}, D) \geq \mathsf{area}(\chi_{\mathbf{C}^*}, S) > \alpha$$

where the last inequality follows from our assumption. This, however, is a contradiction.

Since $\mathsf{maxareas}(\chi_{\mathbf{N}}, \theta + d) \leq \alpha$ implies the that for all $\mathbf{C} \subseteq \mathbf{N}$ such that $|\mathbf{C}| \geq \mathbf{N} - d$, $\mathsf{maxareas}(\chi_{\mathbf{C}}, \theta) \leq \alpha$, if the former occurs with probability at least $1 - \beta$ then so does the latter.

■

Stability of Chord. We now turn to proving the stability of Chord.

Theorem 7.2. *For all* $\theta \leq |\mathbf{N}|/e - d$*, transient Chord is* $(\varepsilon, \delta, \theta)$*-stable for*

$$\varepsilon = \frac{6(\theta + d)}{|\mathbf{N}|} \log\left(\frac{|\mathbf{N}|}{(\theta + d)}\right)\left(1 + \frac{2|\mathbf{N}| \log |\mathbf{N}|}{|\mathbf{A}|}\right), \quad \text{and} \quad \delta = \frac{1}{|\mathbf{N}|^2} + (e^{-\sqrt{|\mathbf{N}|}} \cdot \log |\mathbf{N}|)$$

Proof. From Lemma 7.1, we know that if

$$\Pr[\,\mathsf{maxareas}(\chi_{\mathbf{N}}, \theta + d) \leq \alpha\,] \geq 1 - \beta$$

then

$$\Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\mathbf{N}-d} \mathsf{maxareas}(\chi_{\mathbf{C}}, \theta) \leq \alpha\right] \geq 1 - \beta.$$

But from Corollary 5.4, we have that for $\theta + d \leq |\mathbf{N}|/e$,

$$\alpha = \frac{6|\mathbf{A}|(\theta+d)}{|\mathbf{N}|}\log\left(\frac{|\mathbf{N}|}{(\theta+d)}\right), \qquad \beta = \frac{1}{|\mathbf{N}|^2} + (e^{-\sqrt{|\mathbf{N}|}} \cdot \log|\mathbf{N}|)$$

Finally by applying Theorem 5.5, we conclude that for $\theta + d \leq |\mathbf{N}|/e$,

$$\Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\mathbf{N}-d} (H_1, \mathbf{C}) \text{ is } (\varepsilon,\theta)\text{-balanced}\right] \geq 1-\delta,$$

where ε and δ are as defined in theorem statement.

■

Security of the Chord-based EDHT. In the following Corollary we formally state the security of the standard EDHT when its underlying DHT is instantiated with transient Chord.

Corollary 7.3. *If $|\mathbf{L}| = \Theta(2^k)$, $|I| \leq |\mathbf{N}|/e - d$, and if EDHT^+ is instantiated with Chord, then it is $\mathcal{L}_\varepsilon$-secure with probability at least $1 - 1/|\mathbf{N}|^2 - (e^{-\sqrt{|\mathbf{N}|}} \cdot \log|\mathbf{N}|) - \mathsf{negl}(k)$ in the random oracle model, where*

$$\varepsilon = \frac{6(|I|+d)}{|\mathbf{N}|}\log\left(\frac{|\mathbf{N}|}{(|I|+d)}\right)\left(1 + \frac{2|\mathbf{N}|\log|\mathbf{N}|}{|\mathbf{A}|}\right)$$

The corollary follows from Theorems 6.2 and 7.2 and the fact that Chord has non-committing allocations when H_2 is modeled as a random oracle. The proof is the same as the proof of Corollary 5.7.

Practical considerations. Similar to the discussion following Corollary 5.7 in the perpetual setting, if we set $|I| + d \leq |\mathbf{N}|/\alpha$, where $\alpha \geq e$, then, in expectation, the adversary will learn the query equality leakage of an $O(\log(\alpha)/\alpha)$ fraction of the queries executed between any two churn operations. One thing to notice here is that the inequality $|I| + d \leq |\mathbf{N}|/e$ implies that $|I| + |\mathbf{N}| - |\mathbf{C}| \leq |\mathbf{N}|/e$ which, in turn, implies that $|\mathbf{C}| \geq ((e-1)/e)|\mathbf{N}| + |I|$. Concretely, this means that at all times, the network must have at least $(e-1)/e)|\mathbf{N}| + |I|$ nodes which bounds how many nodes can ever leave the system.

7.1.2 Approach #2: Achieving an Overwhelming Bound on Simulation Success

We now analyze our second strategy which yields an overwhelming bound on the simulation's success probability. As discussed above, we do this by using a new overlay algorithm $\overline{\mathsf{Overlay}}$, which amplifies the probability that $\mathsf{Overlay}$ outputs a good hash function. $\overline{\mathsf{Overlay}}$ takes as input an integer $n \geq 1$ and the security parameter k and chooses a hash function by executing $H_1 \leftarrow \mathsf{DHT.Overlay}(n)$ and checking whether $\mathsf{maxareas}(\chi_\mathbf{N}, 1) \leq (1+c_1)|\mathbf{A}|\log|\mathbf{N}|/|\mathbf{N}|$, where $\chi_\mathbf{N} = (H_1, \mathbf{N})$. If so, it outputs H_1 otherwise it retries for a maximum $r = k/(c_1 \log|\mathbf{N}|)$ times, in which case it fails.

Lemma 7.4. *Let H_1 be the hash function output by $\overline{\mathsf{Overlay}}$ and let $\chi_\mathbf{N} = (H_1, \mathbf{N})$ be the configuration induced by H_1. Then,*

$$\Pr\left[\mathsf{maxareas}(\chi_\mathbf{N}, 1) \leq \frac{(1+c_1)|\mathbf{A}|\log|\mathbf{N}|}{|\mathbf{N}|}\right] \geq 1 - \mathsf{negl}(k),$$

where the probability is over the coins of $\overline{\mathsf{Overlay}}$.

Proof. We call an H_1 bad if $\mathsf{maxareas}(\chi_{\mathbf{N}}, 1)$ is greater than $(1 + c_1)|\mathbf{A}| \log |\mathbf{N}|/|\mathbf{N}|$. Let $\mathcal{E}_i$ be the event that a bad H_1 is sampled in the i-th trial. Then the failure probability of $\overline{\mathsf{Overlay}}$ (i.e., of getting a bad H_1 at the end of $\overline{\mathsf{Overlay}}$) is:

$$\Pr\left[\bigwedge_{1\le i\le r} \mathcal{E}_i\right] \le \frac{1}{|\mathbf{N}|^{c_1 r}} = \frac{1}{e^{c_1 r \log |\mathbf{N}|}} = \frac{1}{e^k} = \mathsf{negl}(k),$$

where the first inequality follows from Lemma 5.2 and from the fact that the $\mathcal{E}_i$'s are independent, and the last equality follows by setting $r = k/(c_1 \log |\mathbf{N}|)$.

■

Corollary 7.5. *Let H_1 be the hash function output by $\overline{\mathsf{Overlay}}$ and let $\chi_{\mathbf{N}} = (H_1, \mathbf{N})$ be the configuration induced by H_1. If $|\mathbf{N}| = \Omega(k)$ and $\theta \le |\mathbf{C}|/e$,*

$$\Pr\left[\mathsf{maxareas}(\chi_{\mathbf{N}}, \theta) \le \frac{6|\mathbf{A}|\theta}{|\mathbf{N}|} \log \frac{|\mathbf{N}|}{\theta}\right] \ge 1 - \mathsf{negl}(k).$$

The proof is similar to the proof of Corollary 5.4. The difference is that the probability p_1 that $\mathsf{maxareas}(\chi_{\mathbf{N}}, 1)$ is bounded by $(3|\mathbf{A}| \log |\mathbf{N}|)/|\mathbf{N}|$ is at most $\mathsf{negl}(k)$ (from Lemma 7.4). The Corollary follows by setting $p_1 = \mathsf{negl}(k)$ and $|\mathbf{N}| = \Omega(k)$.

Stability and security. We now turn to the stability and the security of the Chord-based EDHT.

Theorem 7.6. *If $|\mathbf{N}| = \Omega(k)$ and $\theta \le |\mathbf{N}|/e - d$, transient Chord is $(\varepsilon, \delta, \theta)$-stable for*

$$\varepsilon = \frac{6(\theta + d)}{|\mathbf{N}|} \log\left(\frac{|\mathbf{N}|}{(\theta + d)}\right)\left(1 + \frac{2|\mathbf{N}| \log |\mathbf{N}|}{|\mathbf{A}|}\right), \quad \text{and} \quad \delta = \mathsf{negl}(k)$$

The proof is exactly same as the proof of Theorem 7.2 with the exception that that we use Corollary 7.5. to compute β instead of using Corollary 5.4.

Corollary 7.7. *If $|\mathbf{L}| = \Theta(2^k)$, $|\mathbf{N}| = \Omega(k)$ and $|I| \le |\mathbf{N}|/e - d$, and if EDHT^+ is instantiated with Chord, then it is $\mathcal{L}_\varepsilon$-secure with probability at least $1 - \mathsf{negl}(k)$ in the random oracle model, where*

$$\varepsilon = \frac{6(|I| + d)}{|\mathbf{N}|} \log\left(\frac{|\mathbf{N}|}{(|I| + d)}\right)\left(1 + \frac{2|\mathbf{N}| \log |\mathbf{N}|}{|\mathbf{A}|}\right)$$

Efficiency of $\overline{\mathsf{Overlay}}$. Let $\alpha = ((1+c_1)|\mathbf{A}| \log |\mathbf{N}|)/|\mathbf{N}|$. For each sampled hash function, $\overline{\mathsf{Overlay}}$ checks whether $\mathsf{maxareas}(\chi_{\mathbf{N}}, 1) \le \alpha$. To do this, it computes $H_1(N)$ for all $N \in \mathbf{N}$, sorts all the $H_1(N)$'s to construct $\chi_{\mathbf{N}}$ and, for all $N \in \mathbf{N}$, checks if $\mathsf{area}(\chi_{\mathbf{N}}, N) \le \alpha$. Sorting is $O(|\mathbf{N}| \log |\mathbf{N}|)$ while the remaining steps are $O(|\mathbf{N}|)$. Moreover, $\overline{\mathsf{Overlay}}$ takes a maximum of $k/(c_1 \log |\mathbf{N}|)$ samples so its total running time is $O(k|\mathbf{N}|)$.

References

[1] Bittorrent. https://www.bittorrent.com/.

[2] A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper.

[3] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[4] John W Byers, Jeffrey Considine, and Michael Mitzenmacher. Geometric generalizations of the power of two choices. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 54–63. ACM, 2004.

[5] R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1), 2000.

[6] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security Symposium (NDSS '14)*, 2014.

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2): 4, 2008.

[8] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT '10*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.

[9] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security (CCS '06)*, pages 79–88. ACM, 2006.

[10] Frank Dabek, M Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 202–215. ACM, 2001.

[11] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS operating systems review*, volume 41, pages 205–220. ACM, 2007.

[12] Peter Druschel and Antony Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 75–80. IEEE, 2001.

[13] Michael J Freedman, Eric Freudenthal, and David Mazieres. Democratizing content publication with coral. In *NSDI*, volume 4, pages 18–18, 2004.

[14] M Frans Kaashoek and David R Karger. Koorde: A simple degree-optimal distributed hash table. In *International Workshop on Peer-to-Peer Systems*, pages 98–107. Springer, 2003.

[15] S. Kamara and T. Moataz. Computationally volume-hiding structured encryption. In *Advances in Cryptology - Eurocrypt' 19*, 2019.

[16] Seny Kamara, Tarik Moataz, and Olya Ohrimenko. Structured encryption and leakae suppression. In *Advances in Cryptology - CRYPTO '18*, 2018.

[17] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

[18] Protocol Labs. Filecoin: A decentralized storage network. https://filecoin.io/filecoin.pdf.

[19] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[20] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[21] Athicha Muthitacharoen, Robert Morris, Thomer M Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. *ACM SIGOPS Operating Systems Review*, 36(SI):31–44, 2002.

[22] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[23] Jeffrey Pang, Phillip B Gibbons, Michael Kaminsky, Srinivasan Seshan, and Haifeng Yu. Defragmenting dht-based distributed file systems. In *Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on*, pages 14–14. IEEE, 2007.

[24] Bruno Produit. Using blockchain technology in distributed storage systems. 2018.

[25] Swaminathan Sivasubramanian. Amazon dynamodb: a seamlessly scalable non-relational database service. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 729–730. ACM, 2012.

[26] Moritz Steiner, Damiano Carra, and Ernst W Biersack. Faster content access in kad. In *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on*, pages 195–204. IEEE, 2008.

[27] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[28] Roshan Sumbaly, Jay Kreps, Lei Gao, Alex Feinberg, Chinmay Soman, and Sam Shah. Serving large-scale batch computed data with project voldemort. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, pages 18–18. USENIX Association, 2012.

[29] Basho Technologies. Riak. https://docs.basho.com/riak/kv/2.2.2/learn/dynamo/.

[30] Xiaoming Wang and Dmitri Loguinov. Load-balancing performance of consistent hashing: asymptotic analysis of random node join. *IEEE/ACM Transactions on Networking (TON)*, 15(4):892–905, 2007.

[31] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[32] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015.

A Further Improving Leakage

Recall from the previous sections that the leakage parameter ε is a function of θ, where θ represents the *maximum* number of corruptions in $\mathbf{N}$. Due to this, irrespective of the current overlay (configuration for Chord), we end up leaking the query equality to the adversary with probability proportional to θ, when in fact the number of corruptions in the *current* overlay might be much smaller than θ.

In this section, we develop new general-purpose machinery that leverages this observation to improve the leakage of the standard EDHT: instead of assuming the worst case, and leaking proportional to the maximum number of corruptions θ in $\mathbf{N}$, it leaks proportional to the number of corruptions in $\mathbf{C}$. Therefore instead of having a single leakage parameter ε, we now have a tuple of leakage parameters $\overline{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_\theta)$, where ε_i corresponds to i corruptions.

However, to be able to leak with probability ε_i in an overlay $(\omega, \mathbf{C})$, one also needs to show that $(\omega, \mathbf{C})$ is (ε_i, i)-balanced. Moreover, since the number of corruptions in $(\omega, \mathbf{C})$ is not fixed in advance and depends on the set I (the nodes that the adversary corrupts), one needs to show that $(\omega, \mathbf{C})$ is (ε_i, i)-balanced for all $1 \leq i \leq \theta$. We name this stronger notion of balance as *strong stability*. At a high level, a transient DHT is strongly stable if with probability atleast $1 - \delta$, all large enough $\mathbf{C}$ are (ε_i, i)-balanced, for all $1 \leq i \leq \theta$. Once we have shown our DHT to be strongly stable, then depending on the set I of corrupted nodes, one can compute $i = |I \cap \mathbf{C}|$ and leak according to ε_i.

Definition A.1 (Strong stability)**.** *We say that a transient distributed hash table is $(\overline{\varepsilon}, \delta, \theta)$-strongly-stable, if*

$$\Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta'\geq\theta} \bigwedge_{1\leq i\leq\theta} (\omega, \mathbf{C}) \text{ is } (\varepsilon_i, i)\text{-balanced}\right] \geq 1 - \delta,$$

where the probability is over the choice of ω.

A.1 Security

In this section, we describe at a high level our improved leakage function and state the security of the transient EDHT when it is instantiated with a strongly stable DHT.

The new leakage profile. The leakage function is similar to the leakage function of the transient EDHT described in Section 6. The only difference is that it now samples the bit $b_\ell = 1$ with probability ε_i where i is the number of corruptions in $\mathbf{C}$. More precisely in Step 1a, it does the following:

$$b_\ell \leftarrow \mathsf{Ber}(\varepsilon_i), \text{ where } i = |\mathbf{C} \cap I|.$$

We now show that EDHT^+ is $\mathcal{L}_{\overline{\varepsilon}}$-secure in the transient setting with probability negligibly close to $1 - \delta$ when its underlying transient DHT is $(\overline{\varepsilon}, \delta, \theta)$-strongly-stable and is non-committing.

Theorem A.2. *If $|I| \leq \theta$ and DHT^+ is $(\overline{\varepsilon}, \delta, \theta)$-strongly-stable and has non-committing and label-independent allocations, then EDHT^+ is $\mathcal{L}_{\overline{\varepsilon}}$-secure with probability at least $1 - \delta - \mathsf{negl}(k)$.*

We skip the proof as it is exactly same as the proof of Theorem 6.2.

A.2 Analysis of Chord

Here we analyse the strong stability of Chord. The high level idea is the following. Since a bound on $\mathsf{maxareas}(\chi_{\mathbf{N}}, i+d)$ implies a bound on $\mathsf{maxareas}(\chi_{\mathbf{C}}, i)$, we need to simply bound the former, which can then be translated into ε_i. Then to compute δ, we need to bound the probability that for all i, $\mathsf{maxareas}(\chi_{\mathbf{N}}, i+d)$ is bounded, and we do it by applying a union bound on the individual probabilities of $\mathsf{maxareas}(\chi_{\mathbf{N}}, i+d)$ being bounded.

Theorem A.3. *For $|\mathbf{N}| = \Omega(k)$ and all $\theta \leq |\mathbf{N}|/e - d$, transient Chord is $(\bar{\varepsilon}, \delta, \theta)$-strongly-stable for*

$$\varepsilon_i = \frac{6(i+d)}{|\mathbf{N}|} \log\left(\frac{|\mathbf{N}|}{(i+d)}\right)\left(1 + \frac{2|\mathbf{N}|\log|\mathbf{N}|}{|\mathbf{A}|}\right), \quad \text{and} \quad \delta = \mathsf{negl}(k)$$

Proof. It follows from the union bound that if for $i \leq \theta$

$$\Pr\left[\, \mathsf{maxareas}(\chi_{\mathbf{N}}, i+d) \leq \alpha_i \,\right] \geq 1 - \beta_i \tag{13}$$

then,

$$\Pr\left[\bigwedge_{i \leq \theta} \mathsf{maxareas}(\chi_{\mathbf{N}}, i+d) \leq \alpha_i\right] \geq 1 - \sum_{i \leq \theta} \beta_i \tag{14}$$

But if Equation 14 holds, then so does the following by Lemma 7.1

$$\Pr\left[\bigwedge_{i \leq \theta} \bigwedge_{\mathbf{C} \subseteq \mathbf{N} : |\mathbf{C}| \geq \mathbf{N} - d} \mathsf{maxareas}(\chi_{\mathbf{C}}, i) \leq \alpha_i\right] \geq 1 - \sum_{i \leq \theta} \beta_i \tag{15}$$

Now by using Corollary 7.5 on Equation 13, we know that for $i + d \leq |\mathbf{N}|/e$, and $|\mathbf{N}| = \Omega(k)$,

$$\Pr\left[\, \mathsf{maxareas}(\chi_{\mathbf{N}}, i+d) \leq \alpha_i \,\right] \geq 1 - \beta_i$$

for

$$\alpha_i = \frac{6|\mathbf{A}|(i+d)}{|\mathbf{N}|} \log\left(\frac{|\mathbf{N}|}{(i+d)}\right), \qquad \beta_i = \mathsf{negl}(k)$$

Finally by applying Theorem 5.5 on Equation 15, we conclude that for $i+d \leq |\mathbf{N}|/e$ and $|\mathbf{N}| = \Omega(k)$,

$$\Pr\left[\bigwedge_{i \leq \theta} \bigwedge_{\mathbf{C} \subseteq \mathbf{N} : |\mathbf{C}| \geq \mathbf{N} - d} (H_1, \mathbf{C}) \text{ is } (\varepsilon_i, i)\text{-balanced}\right] \geq 1 - \delta,$$

where ε and δ are as defined in theorem statement.

■

Corollary A.4. *If $|\mathbf{L}| = \Theta(2^k)$, $|\mathbf{N}| = \Omega(k)$ and $|I| \leq |\mathbf{N}|/e - d$, and if EDHT^+ is instantiated with Chord, then it is $\mathcal{L}_{\bar{\varepsilon}}$-secure with probability at least $1 - \mathsf{negl}(k)$ in the random oracle model, where*

$$\varepsilon_i = \frac{6(i+d)}{|\mathbf{N}|} \log\left(\frac{|\mathbf{N}|}{(i+d)}\right)\left(1 + \frac{2|\mathbf{N}|\log|\mathbf{N}|}{|\mathbf{A}|}\right)$$