Encrypted Distributed Dictionaries

Archita Agarwal[*]
MongoDB

Seny Kamara[†]
MongoDB and Brown University

Abstract

End-to-end encrypted databases have been heavily studied in the last two decades. A crucial aspect that previous work has neglected, however, is that real-world databases are distributed in the sense that data is partitioned among a cluster of nodes—as opposed to being stored on a single node. In this work, we initiate the study of encrypted *distributed* data structures which are end-to-end encrypted variants of distributed data structures; themselves fundamental to the design of distributed databases. In particular, we design and analyze encrypted variants of distributed dictionaries (EDDX), which are an important building block in distributed system design and have applications ranging from content delivery networks to off-chain storage networks for blockchains and smart contracts.

We formalize the notion of an encrypted DDX and provide simulation-based security definitions that capture the security properties one would desire from such an object. We propose an EDDX construction that uses a distributed hash table (DHT) as a black box. Interestingly, we show that our construction 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 our construction, a plaintext DHT needs to satisfy a form of "programmability", a property that usually only emerges in context of cryptographic primitives. To show that these properties are indeed achievable in practice, we study the balancing properties of the Chord DHT—arguably one of the most influential DHT—and show that it is also programmable. Finally, we consider the problem of encrypted DDXs in the context of transient networks, where nodes can be arbitrarily added or removed from the network.

[*] `archita_agarwal@alumni.brown.edu`. Work done while at Brown University.

[†] `seny@brown.edu`

Contents

1 Introduction

As we continue to produce and consume large amounts of sensitive and intrusive data, we are faced with the problem of securing it. With constant data breaches, it is clear that the traditional database management systems are not enough to protect the data stored in them. They sometimes encrypt data at rest and in transit, but the security provided is still piece-meal. In practice, they decrypt the data before use and each decryption exposes the data and increases its likelihood of being stolen.

An alternative approach is end-to-end encryption, where data is kept encrypted at all times. This approach provides much stronger security guarantees than in-transit and at-rest encryption. Given the high level of interest in end-to-end encrypted database solutions, they have been widely researched in the past two decades, and researchers have developed a multitude of solutions.

Distributed databases. One crucial point that the research community has missed, however, is that in reality databases are *distributed* in the sense thatthe data is not stored on a single machine but instead is partitioned amongst a cluster of machines. Queries to the database are routed to the right set of machines in the cluster, and results from the individual machines are merged before being returned back to the user. This kind of distributed database architecture allows large internet companies like Amazon, Google and Facebook to scale and provide services using commodity hardware in data centers distributed across the world.

In order to develop real-world usable encrypted databases, it is crucial that we design solutions keeping the distributed nature of databases in mind. Developing ad-hoc cryptographic solutions for distributed systems might not be the best for security and efficiency, and therefore it is important that we start reasoning about the two together. In this work, we initiate the study of *encrypted distributed data structures*. In particular, we consider the case of *encrypted distributed dictionaries*. While dictionaries are an important building block of encrypted data structures and encrypted databases, they are also important on their own since they capture NoSQL databases like key-value stores, which are popular in industry due to their efficiency guarantees. For example, Amazon's DynamoDB [16, 37] underlies the Amazon shopping cart, LinkedIn's Voldemort[41], Facebook's Cassandra [29], or Google's BigTable [12].

Distributed dictionaries and hash tables. The most fundamental building block in the design of highly scalable and reliable distributed databases are *distributed dictionaries* (DDX). Roughly speaking, a DDX is a 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. DDXs are distributed in the sense that the pairs are stored by a set of n nodes $N_1, \ldots, N_n$, and they provide many useful properties; the most important of which are load balancing and fast data retrieval and storage even when the data is distributed across multiple nodes. In this work, we focus on two settings, the perpetual setting where the set of nodes underlying the dictionary is fixed, and the transient setting where nodes can be added/removed.

One of the most common ways of instantiating a DDX is through a distributed hash table (DHT). To communicate and route messages to and from nodes, DHTs rely on (1) a randomly generated *overlay network* which, intuitively, arranges nodes in a chosen network topology (e.g., star topology, tree topology) and (2) a distributed routing protocol that routes messages between nodes. DHTs were first introduced in the context of peer-to-peer (P2P) file sharing but have since found applications beyond P2P systems; including for load balancing, distributed storage and blockchains. An important limitation of early DHTs was that get and put operations were supported in $O(n)$, where n is the number of nodes in the system. This was improved by a new generation of DHTs for "structured P2P" settings like Chord [40] and Pastry [36] which support gets and puts in $O(\log n)$.

Applications of DDXs/DHTs. It is hard to overstate the impact that DDXs 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 [27] 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 [20, 39]. DHTs are also used by many P2P systems like BitTorrent [2] and its many trackerless clients including Vuze, rTorrent, Ktorrent and μTorrent. Many distributed file systems are built on top of DHTs, including CFS [15], Ivy [32], Pond [34], PAST [18].

Currently, the field of distributed systems is going through revolution brought about by the introduction of blockchains [33]. 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 [42, 3], Enigma [43], Storj [35] and Filecoin [28] rely on off-chain storage: Ethereum, Enigma and Storj on their own custom networks and Filecoin on IPFS [4]. Due to the storage and scalability requirements of these blockchains, these off-chain storage networks often use DHTs as a core building block.

EDDXs and structured encryption. Due to the ubiquity of DDXs, we believe that a formal study of confidentiality in DDXs is a well-motivated problem of practical importance. In this work, we introduce the notion of an *encrypted DDX* (EDDX) and propose formal syntax and security definitions for these objects. Designing the secure version of such a basic building block will enable us to build a range of privacy-preserving systems on top of them.

The notion of an EDDX 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, EDDXs 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. Due to this natural connection, we formalize EDDX schemes using STE-style syntax and security definitions.

Even though encrypted DDXs syntactically look similar to encrypted dictionaries (in context of structured encryption), we will see that defining and analyzing the security of EDDXs is significantly more complex. This is because we will consider an adversary that corrupts a fraction of nodes in the network, and it is not obvious how to precisely analyze the security one can achieve against an adversary with only partial view of the data. To illustrate this point, suppose a subset of nodes are corrupted and collude. During the operation what information can they learn about the 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, 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 for efficieny reasons, DDXs 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 DDXs and the many systems they support. We define formal syntax and security definitions of an EDDX scheme and equipped with these definitions, we design and analyze a concrete EDDX construction. We make several contributions.

A DHT-based EDDX scheme. Our EDDX scheme BDX uses a DHT as a building box. Given a label/value pair (ℓ, v), the client computes $(F_{K_1}(\ell), \mathsf{Enc}_{K_2}(v))$, where F is a pseudo-random function and Enc is a symmetric-key encryption scheme, and then it simply sends the "encrypted" pair to the underlying DHT to store. The DHT then assigns this "encrypted" pair to a storage node in a load balanced manner, handles routing, and moves pairs around the network if a node is added or removed from the network. As we will see, analyzing and proving the security of even this simple scheme is complex. The reason is because, as we will see, the security of the BDX is tightly coupled with how the underlying DHT is designed.

Formalizing DHTs. To better understand the security properties of BDX, we will isolate properties of its underlying DHT 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. 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. This abstraction will allow us to analyze various properties of DHTs and prove bounds on their behavior. Furthermore, it lays the foundations that will allow future work to concretely analyze the security of BDX when it is instantiated with the myriad of DHT designs proposed in the literature [30, 21].

As mentioned above, we found that the security of BDX 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). Another interesting finding we made was that if BDX is to satisfy our simulation-based definition, then the underlying DHT has to satisfy a form of "programmability". 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 node. We found the appearance of programmability in the context of DHTs quite surprising as it is usually a property that comes up in the context of cryptographic primitives.

Having isolated the properties we need from a DHT in order to prove the security of BDX, 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 [27]—which is arguably the most influential DHT—is both balanced and non-committing in the sense that it supports the kind of programmability discussed above. Without getting into details of how this DHT works, we mention here that it make use of two hash functions, and we show that it is both balanced and non-committing if one of its hash function is modeled as a random oracle.

Security of EDDXs. Another contribution we make is a simulation-based definition of security for EDDXs. The definition is in the real/ideal-world paradigm commonly used to formalize the security of multi-party computation [8]. 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 EDDXs 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 EDDX 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 a functionality that captures the ideal security properties of EDDXs. As discussed above, for any EDDX scheme, including BDX, 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.

EDDX and structured encryption. STE schemes are encryption schemes that encrypt data structures in such a way that they can be privately queried. Encrypted dictionaries (in context of structured encryption) are captured as a special case of our work where the network consists of a single node that is corrupted. We note that this connection to single-node encrypted dictionaries is not just syntactical, but also holds with respect to the security definitions of both objects and to their leakage profiles. Indeed the leakage profile of BDX on a single-node network reduces to the leakage profile of common dictionary encryption schemes [13, 11]. This leakage, however, represents the "worst-case" leakage of BDX. This is due to the fact that BDX leaks the operation equality, opeq of labels probabilistically whereas standard single-node encrypted dictionaries leak it for all the labels. 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 [26, 24]. That is, instead of (or in addition to) compiling STE schemes as in [26] or of transforming the underlying data structures as in [24], one could *distribute* the encrypted data structure.

Probabilistic leakage. Our security definition allows us to formally study any leakage produced by EDDX schemes. Interestingly, our analysis of BDX will show that it achieves a very novel kind of leakage profile, which in 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 first time such a leakage profile has been encountered. Here, the information it leaks (when it does leak) is the operation equality pattern (see [26] for a discussion of various leakage patterns) which reveals if and when an operation on the same label 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, its load balancing properties. Specifically, the better the DHT load balances its data the smaller the probability that BDX will leak the operation 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.

Transient EDDXs. All the analysis discussed above was for what we call the *perpetual* setting where the set of nodes in the network is fixed. Note that the perpetual setting is realistic and interesting in itself. It captures, for example, how DDXs 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 clients can arbitrarily add or remove nodes from the network. We extend our syntax and security definitions to the transient setting, and prove that our extended construction BDX^+—equipped with certain add and remove node protocols—achieves another probabilistic leakage profile. 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.

Having analyzed BDX^+ 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.

1.2 Related Work

Since we already discussed related work on DDXs/DHTs and their applications, in this section we only focus on the work related in encrypted search literature.

Encrypted search was first considered explicitly by Song, Wagner and Perrig in [38] which introduced the notion of searchable symmetric encryption (SSE). Curtmola, Garay, Kamara and Ostrovsky introduced and formulated the notion of adaptive semantic security for SSE [14] together with the first sub-linear and optimal-time constructions. Chase and Kamara introduced the notion of structured encryption (STE) which generalizes SSE to arbitrary data structures [13]. Multi-map encryption schemes are a special case of STE and have been used to achieve optimal-time single-keyword SSE [14, 25, 11, 5, 6], sub-linear Boolean SSE [10, 22], encrypted range search [19, 17], encrypted relational databases [23], and graph encryption [13, 31]. Some of these techniques have been deployed in real-world databases as well, e.g., MongoDB's queryable encryption now offers its clients the ability to encrypt and query their data [1].

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 $x \overset{\$}{\leftarrow} X$ 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 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.

3 Encrypted Distributed Dictionaries

An encrypted distributed dictionary scheme encrypts a distributed dictionary in such a way that it can support efficient get and put operations on encypted data stored across multiple nodes. We formalize two types of schemes depending on whether or not a client can add/remove nodes from the underlying storage network. The perpetual setting comprises of a fixed set of nodes that are all known at the initialization time, whereas in the transient setting the nodes are not known a-priori, and the client and add and remove nodes at any time. The former is suitable for settings where a client rents a fixed sized cluster from a cloud provider like AWS or Azure, while the latter is more suitable for settings where a client adds/removes nodes to its rented cluster depending on its current storage needs.

Definition 3.1 (Perpetual EDDX scheme)**.** *A perpetual* EDDX *scheme* $\Sigma_{\mathsf{EDDX}} = (\mathsf{Init}, \mathsf{Put}, \mathsf{Get})$ *is a collection of three protocols between the client* $\mathcal{C}$ *and a set of currently active nodes* $\mathbf{C} = (N_1, \ldots, N_n)$*. They work as follows:*

- $(K; \mathsf{EDDX}_1; \ldots; \mathsf{EDDX}_n) \leftarrow \mathsf{Init}_{\mathcal{C},\mathbf{C}}(1^k; \perp; \ldots; \perp_n)$ *is a probabilistic protocol where the client enters the security parameter, while the nodes input nothing. The client receives a key* K *while a node* N_i *receives a shard* EDDX_i *of the encrypted distributed dictionary* EDDX*.*

- $(\perp; \mathsf{EDDX}'_1; \ldots; \mathsf{EDDX}'_n) \leftarrow \mathsf{Put}_{\mathcal{C},\mathbf{C}}(K, \ell, v; \mathsf{EDDX}_1; \ldots; \mathsf{EDDX}_n)$ *is a (probabilistic) protocol, where the client inputs the secret key* K *and a label/value pair* ℓ *and* v*, while the nodes input their respective shards of the encrypted distributed dictionary. At the end of the protocol, the client receives nothing while the nodes receive updated shards of the encrypted distributed dictionary.*

- $(v; \bot_1; \ldots; \bot_n) \leftarrow \mathsf{Get}_{\mathcal{C},\mathbf{C}}(K, \ell; \mathsf{EDDX}_1; \ldots; \mathsf{EDDX}_n)$ *is same as the put protocol with the difference that the client receives a value v, while the nodes receive nothing at the end of the protocol.*

Definition 3.2 (Transient EDDX scheme)**.** *A transient* EDDX *scheme* $\Sigma_{\mathsf{EDDX}^+} = (\mathsf{Init}, \mathsf{Put}, \mathsf{Get}, \mathsf{AddNode}, \mathsf{RemoveNode})$ *is a collection of five protocols where the first three are same as in the perpetual setting and the last two work as follows:*

- $(\bot; \mathsf{EDDX}'_1; \ldots; \mathsf{EDDX}'_{n+1}) \leftarrow \mathsf{AddNode}_{\mathcal{C},\mathbf{C},N^+}(\bot; \mathsf{EDDX})$ *is a (probabilistic) protocol executed when the client wants to add a node N^+ not already active to the network. The client inputs nothing while the original nodes input their respective shards EDDX_i. At the end of the protocol, the nodes including the node N^+ receive an updated shard EDDX'_i of the encrypted distributed dictionary.*
- $(\bot; \mathsf{EDDX}'_1; \ldots; \mathsf{EDDX}'_{n-1}) \leftarrow \mathsf{RemoveNode}_{\mathcal{C},\mathbf{C}}(\bot; \mathsf{EDDX})$ *is same as the* AddNode *protocol and is executed when the client wants to remove an active node $N^- \in \mathbf{C}$ from the network. At the end of the protocol, each of the remaining nodes receive an updated shard.*

Note that when a node $N \in \mathbf{C}$ is removed from the network, the set of active nodes $\mathbf{C}$ automatically shrinks to exclude N. Similarly, when a node $N \notin \mathbf{C}$ is added to 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. Also note that here the client initiates the addition/removal of nodes, but we can easily adapt our definitions to include another entity, e.g. cluster manager, to initiate these processes instead of the client.

Security in the perpetual setting. We now turn to formalizing the security of an EDDX scheme. We do this by combining the definitional approaches used in secure multi-party computation [8] and in structured encryption [14, 13]. 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 an 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.

The real-world experiment. The experiment is executed between a client $\mathcal{C}$, a set $\mathbf{C}$ of n nodes $N_1, \ldots, N_n$, an environment $\mathcal{Z}$ and an adversary $\mathcal{A}$. 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 and the nodes then execute $(K; \mathsf{EDDX}_1; \ldots; \mathsf{EDDX}_n) \leftarrow \mathsf{Init}(1^k; \bot; \ldots; \bot_n)$ protocol, and the client receives a secret key K while a node N_i recieves a shard EDDX_i of the encrypted distributed dictionary EDDX. $\mathcal{Z}$ then adaptively chooses a polynomial number of operations op_j, where $\mathsf{op}_j \in \{\mathtt{get}, \mathtt{put}\} \times \mathbf{L} \times \{\mathbf{V}, \bot\}$ and sends it to $\mathcal{C}$. If $\mathsf{op}_j = (\mathtt{get}, \ell)$, the client $\mathcal{C}$ executes $\mathsf{Get}(K, \ell; \mathsf{EDDX}_1; \ldots; \mathsf{EDDX}_n)$ protocol with the nodes, and if $\mathsf{op}_j = (\mathtt{put}, \ell, v)$, $\mathcal{C}$ initiates $\mathsf{EDDX.Put}(K, \ell, v; \mathsf{EDDX}_1; \ldots; \mathsf{EDDX}_n)$. 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.

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

$\mathcal{F}^{\mathcal{L}}_{\mathsf{DX}}$ stores a dictionary DX initialized to empty, and it proceeds as follows, running with client $\mathcal{C}$, active nodes in $\mathbf{C}$ and a simulator Sim:

- Upon receiving a $(\mathtt{put}, \ell, v)$ message 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.
- Upon receiving a $\mathsf{Get}(\ell)$ message 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.
- Upon receiving a $\mathtt{removenode}(N)$ message from client $\mathbf{C}$, where $N \in \mathbf{C}$, it returns the leakage $\mathcal{L}(\mathsf{DX}, (\mathtt{removenode}, N))$ to the simulator Sim and updates its set $\mathbf{C}$.
- Upon receiving an $\mathtt{addnode}(N)$ message from client $\mathbf{C}$, where $N \in \mathbf{N} \setminus \mathbf{C}$, it returns the leakage $\mathcal{L}(\mathsf{DX}, (\mathtt{addnode}, N))$ to the simulator Sim and updates its set $\mathbf{C}$.

Figure 1: $\mathcal{F}^{\mathcal{L}}_{\mathsf{DX}}$: The transient DX functionality parameterized with leakage function $\mathcal{L}$. In the perpetual setting, the functionality only implements the get and put operations.

The ideal-world experiment. The experiment is executed between a client $\mathcal{C}$, a set $\mathbf{C}$ 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 $\mathcal{F}^{\mathcal{L}}_{\mathsf{DDX}}$ (Figure 1). 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 $\mathcal{F}^{\mathcal{L}}_{\mathsf{DX}}$. If $\mathsf{op}_j = (\mathtt{get}, \ell)$, the functionality executes $\mathcal{F}^{\mathcal{L}}_{\mathsf{DX}}.\mathsf{Get}(\ell)$. Otherwise, if $\mathsf{op}_j = (\mathtt{put}, \ell, v)$ the functionality executes $\mathcal{F}^{\mathcal{L}}_{\mathsf{DX}}.\mathsf{Put}(\ell, v)$. $\mathcal{C}$ 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 3.3 ($\mathcal{L}$-security)**.** *We say that a perpetual encrypted distrubuted dictionary* $\Sigma_{\mathsf{EDDX}} = (\mathsf{Init}, \mathsf{Put}, \mathsf{Get})$ *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 $z \in \{0,1\}^*$,*

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

Security in the transient setting. Note that at any time, an EDDX only has a set $\mathbf{C}$ of active nodes (i.e., the nodes currently in the network). Let $\mathbf{N}$ be the set of all the possible nodes, e.g., one can think of $\mathbf{N}$ as the set of all IPv4 addresses but only a subset $\mathbf{C}$ are in the network.

Then in the transient setting, the real and ideal experiments are the same as the perpetual setting with the following two differences. First, the environment selects and activates a subset a set $\mathbf{C} \subseteq \mathbf{N}$ of nodes in the beginning; second, the environment also sends $\mathsf{AddNode}$ and $\mathsf{RemoveNode}$ operations adaptively along with Get and Put operations to the client. For $\mathsf{AddNode}$ operation, it selects a node $N \in \mathbf{N} \setminus \mathbf{C}$ that is not already in the network and, for $\mathsf{RemoveNode}$ operation, it selects a node $N \in \mathbf{C}$ already in the network.

4 Distributed Hash Tables

A distributed hash table is a distributed storage system that instantiates a distributed dictionary data structure. Our encrypted distributed dictionary scheme BDX uses them as a building block where it encrypts the label/value pairs before storing them in a DHT. In order to help us analyze the

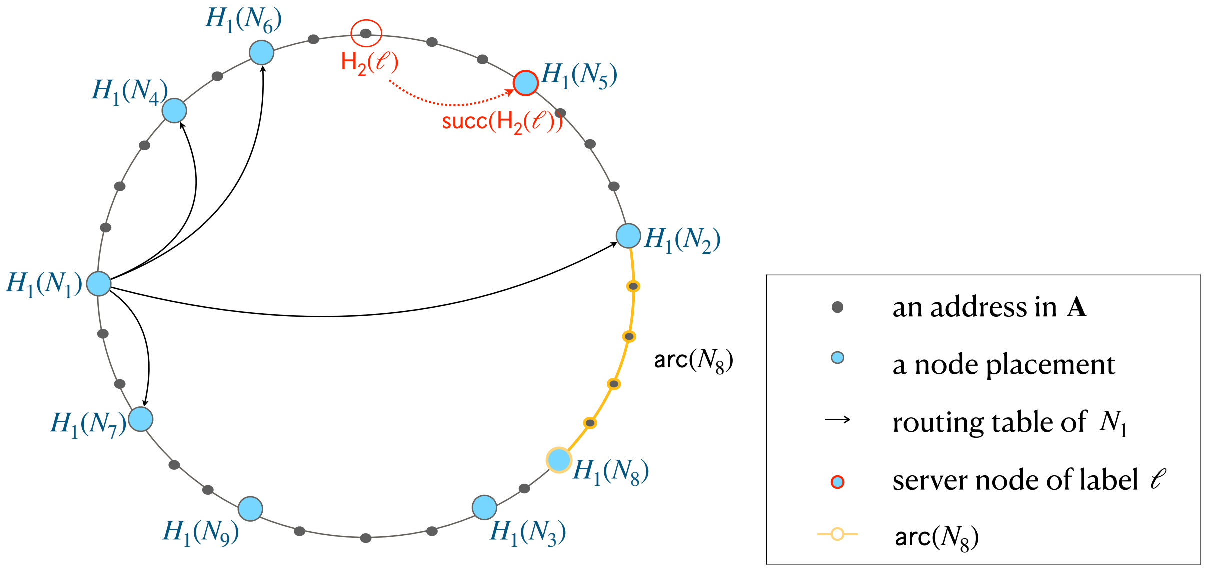

Figure 2: Chord DHT

security of BDX, we formalize DHTs and abstract their core components. The formalism introduced here is in itself interesting and provides us with a framework to model distributed systems. We use Chord DHT as a running example to make the exposition easier to understand.

4.1 The Chord DHT

Setting up Chord. Chord works in a logical m-bit address space $\mathbf{A} = \{0, \ldots, 2^{m-1}\}$. It views the set of addresses to be arranged in a ring (see Fig 2). At the time of setup, it samples two hash functions, H_1 and H_2, where H_1 assigns each node N an address $H_1(N)$ and H_2 assigns each label ℓ an address $H_2(\ell)$. We call the set $\chi = \{H_1(N_1), \ldots, H_1(N_n)\}$ of addresses asssigned to nodes a *configuration*. Intuitively, a configuration denotes the placement of nodes on the ring of addresses.

Given a configuration χ, Chord defines the "successor" of an address a as the node that follows a on the address ring in clockwise direction. The first successor of a is the first node that follows a, the second successor is the second node that follows a, and so on. The predecessors of an address/node are defined in the same way. We use the notation $\mathsf{succ}_\chi(a)$ to denote the first successor of address a, and $\mathsf{pred}_\chi(a)$ to denote its first predecessor. For visual clarity, we sometimes drop χ from the subscript when its clear from context.

Routing in Chord. At the time of setup, each node also constructs a routing table where they store the addresses of their 2^ith successor where $0 \leq i \leq \log n$. 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 given a configuration χ, the route between any two nodes is fixed. Moreover, the structure of routing tables ensures that the route lengths are at most $\log n$ long [40].

Storing and retrieving. Once the DHT is instantiated, each node instantiates an empty dictionary data structure DX_i. When a client executes a Put operation on a label/value pair (ℓ, v), it chooses a node N_s (called the front-end node) and forwards the Put request to N_s. N_s then computes $N_d = \mathsf{succ}(H_2(\ell))$ and uses the Chord routing protocol to send the pair (ℓ, v) to the node N_d who stores it in its local dictionary DX_i. Similarly, when executing a Get query on a label ℓ, the client chooses and forwards its get request to a node N_s. The front-end node N_s then computes $N_d = \mathsf{succ}(H_2(\ell))$ and, uses the Chord routing protocol to send the label ℓ to N_d. The latter looks up ℓ in its local dictionary DX_i and return the associated value v to N_s, which in turn returns v to the client. We note that Chord allows its clients to choose any node as the front-end node. Moreover, it does not restrict them to connect to the same node everytime the client wants to query the same ℓ.

4.2 Formalizing DHTs

Syntax. We formalize perpetual DHTs as a collection of six algorithms $\mathsf{DHT} = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{FrontEnd}, \mathsf{Init}, \mathsf{Put}, \mathsf{Get})$ that work as follows:

- The first three algorithms, $\mathsf{Overlay}$, Alloc and $\mathsf{FrontEnd}$ are executed only once by the entity responsible for setting up the DHT. They all take as input an integer $n \geq 1$. $\mathsf{Overlay}$ outputs a parameter ω from a space Ω, Alloc outputs a parameter ψ from a space Ψ, and $\mathsf{FrontEnd}$ outputs a parameter ϕ from space Φ. We refer to these parameters as the *DHT parameters* and represent them by $\Gamma = (\omega, \psi, \phi)$. Each DHT has an address space $\mathbf{A}$ and the DHT parameters in Γ define different components of the DHT over this address space. For example, ω maps node names to addresses in $\mathbf{A}$, ψ maps labels to addresses in $\mathbf{A}$, ϕ maps operation requests to the address of a front-end node (or starting node). Once the responsible entity generates Γ, it sends it to all the nodes in the network.
- The next algorithm $\mathsf{Init}(\Gamma)$ takes the DHT parameters as input and is executed by each active node $N_i \in \mathbf{C}$ in the network. It outputs an empty dictionary DX_i and a state st_i at node N_i.
- Finally, the last two protocols Put (Get) are executed between the client and the nodes, where the client inputs its label/value pair (a label for Get) and the nodes input their dictionaries and states. At the end of the protocol, the client receives nothing (a value v in case of Get), and the nodes receive updated dictionaries (nothing for for Get).

Abstracting DHTs. We now abstract the core components of DHTs out. These abstractions will be helpful in our security analysis. Given Γ, we can describe a DHT using a tuple of function families $(\mathsf{addr}, \mathsf{server}, \mathsf{route}, \mathsf{fe})$ that are all parameterized by subset of parameters in Γ. These functions are defined as

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

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 node that stores it, route_ω maps two nodes to the set of nodes on the route between them, and fe_ϕ maps labels to node addresses who forward client requests to the rest of the network. 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, these are implemented by the six algorithms defined in the last section.

We also note that at any time, a DHT only has a subset $\mathbf{C} \subseteq \mathbf{N}$ of active nodes (i.e., the nodes currently in the network); e.g., one can think of $\mathbf{N}$ as the set of all IPv4 addresses but only a subset would join the DHT network. Together ω and $\mathbf{C}$ create an overlay, and henceforth we refer to $(\omega, \mathbf{C})$ as an overlay.

Instantiating abstractions for Chord. For Chord, $\omega = H_1$ and $\psi = H_2$, where H_1 and H_2 are modeled as random oracles. Then the overlay $(\omega, \mathbf{C})$ is equivalent to a configuration $\chi = \{H_1(N_1), \ldots, H_1(N_n)\}$. The map addr_ω is H_1 which assigns to each active node $N \in \mathbf{C}$ an address $H_1(N)$ in $\mathbf{A}$. The map $\mathsf{server}_{\omega,\psi}$ is the function $\mathsf{succ} \circ H_2$, that stores a label ℓ at the successor of $H_2(\ell)$. Recall that given a configuration χ, the route between any two nodes is fixed. Therefore, the route_ω map for Chord is deterministic and well defined.

Further recall that Chord allows its clients to choose any node as the front-end node to issue its operations. Moreover, it does not restrict them to connect to the same node $\mathsf{fe}_\phi(\ell)$, time the client wants to query the same ℓ. This means that for Chord, fe_ϕ is not necessarily a function but can be a one-to-many relation. Unfortunately we will later see that we cannot prove that an EDDX based on Chord is "secure" for arbitrary fe_ϕ's. We therefore modify Chord and let ϕ be a third hash function H_3 that maps labels to nodes currently active. Then, fe_ϕ is the hash function H_3 itself that assigns a front-end node $H_3(\ell)$ to each request for ℓ.

Visible addresses. An important notion for our purposes will be that of the set of *visible addresses* to a node. Intuitively, we say that an address a is visible to a node N, if labels mapped to a are either stored by N or are routed by it. Notice that whether or not addresses mapped to a are routed by N depend on the node where the request for the label originates. Changing the frontend node, changes the route, even if the destination node remains the same. Therefore, instead of simply defining the visibility of a node, we define what we call a node's N_s-visibility, where N_s is the starting node of the routes. Throughout we assume the set of visible addresses to be efficiently computable.

Definition 4.1 (N_s-visibility)**.** *Let $(\omega, \mathbf{C})$ be an overlay, ψ be an allocation parameter and $N_s \in \mathbf{C}$ be an active node. Then we say an address $a \in \mathbf{A}$ is N_s-visible to a node $N \in \mathbf{C}$ if there exists a label $\ell \in \mathbf{L}$ such that if ψ allocates ℓ to a, then either:* (1) *$N = \mathsf{server}_{\omega,\psi}(\ell)$; or* (2) *$N \in \mathsf{route}_\omega(N_s, \mathsf{server}_{\omega,\psi}(\ell))$. We denote the set of N_s-visible addresses to N by $\mathsf{Vis}(N_s, N)$.*

Since the set of N_s-visible addresses depends on ω, the set $\mathbf{C}$ of nodes that are currently active, and the allocation parameter ψ, we subscript $\mathsf{Vis}_{\omega,\mathbf{C},\psi}(N_s, N)$ with all these paramters. Finally, we extend the notion of visibility of a single node to the visibility of a set of nodes $S \subseteq \mathbf{C}$. It is defined simply as the union of visibilities of individual nodes, i.e., for $S \subseteq \mathbf{C}$, $\mathsf{Vis}_{\omega,\mathbf{C},\psi}(N_s, S) = \cup_{N \in S} \mathsf{Vis}_{\omega,\mathbf{C},\psi}(N_s, N)$. Again, for visual clarity, we will drop the subscripts wherever they are clear from the context.

Visibility in Chord. Given a configuration χ, let *arc* of a node N is the set of addresses in $\mathbf{A}$ between N and its predecessor in χ (see Figure 2). 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 defined earlier.

Recall that an address a is visible to a node N, if (1) N stores the labels hashed to a or, (2) N routes the labels hashed to a (starting from N_s). In Chord, a node N stores all the labels that are hashed to its arc. Therefore, due to (1), all the addresses in N's arc are visible to it. Moreover, due to (2), addresses in the arcs of any other node N' are also N_s-visible to N, if N falls on the route between N_s and N'. This is because, all the labels hashed in the arc of N' will be routed by N. In summary, given a fixed configuration χ, and two distinct nodes $N_s, N \in \mathbf{C}$:

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

Allocation distribution. The next 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. We therefore simply refer to this distribution as the DHT's allocation distribution. Given an overlay $(\omega, \mathbf{C})$ and a front-end parameter ϕ, we now define two distributions $\Delta_1(S, \ell)$ and $\Delta_2(\ell)$, the first of which is parameterized by a set of addresses $S \subseteq \mathbf{A}$ and a label $\ell \in \mathbf{L}$, while the second is only paramterized by a label $\ell \in \mathbf{L}$. The distributions are over the choice of the allocation parameter ψ and we assume both of them to be efficiently computable.

The first distribution $\Delta_1(S, \ell)$ intuitively captures the conditional probability of ℓ being assigned to a particular address a, given that it is assigned to an address in S. Formally, an address $a \in \mathbf{A}$ has probability mass function

$$f_{\Delta_1(S,\ell)}(a) = \Pr\left[\, \psi(\ell) = a \mid \psi(\ell) \in S \,\right] = \frac{\Pr\left[\, \psi(\ell) = a \wedge \psi(\ell) \in S \,\right]}{\Pr\left[\, \psi(\ell) \in S \,\right]}$$

However, for an address $a \notin S$, the probability is 0. This is because in this case, $\psi(\ell) \notin S$ and therefore the numerator evaluates to 0. Moreover, for $a \in S$, the numerator evaluates to $\Pr\left[\, \psi(\ell) = a \,\right]$ because the event $\{\psi(\ell) = a\}$ implies the event $\{\psi(\ell) \in S\}$.
For the case of Chord, recall that it assigns labels to addresses using a random oracle H_2. Therefore it follows that for all configurations χ, all labels $\ell \in \mathbf{L}$ and all subsets $S \subseteq \mathbf{A}$, and all $a \in S$,

$$f_{\Delta_1(S,\ell)}(a) = \frac{\Pr\left[\, H_2(\ell) = a \,\right]}{\Pr\left[\, H_2(\ell) \in S \,\right]} = \frac{\frac{1}{|\mathbf{A}|}}{\frac{|S|}{|\mathbf{A}|}} = \frac{1}{|S|}.$$

The second distribution $\Delta_2(\ell)$ captures the probability of ℓ being assigned to an address in S. The probability mass function of a set $S \subseteq \mathbf{A}$ is

$$f_{\Delta_2(\ell)}(S) = \Pr\left[\, \psi(\ell) \in S \,\right].$$

For Chord, it is equal to $|S|/|\mathbf{A}|$. We stress that both distributions are over the randomness of the algorithm Alloc.

Non-committing allocations. As we will see in Section 5, our EDDX 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 allocations it produces be non-committing in the sense that it supports a form of equivocation. More precisely, for some fixed overlay $(\omega, \mathbf{C})$ and allocation parameter ψ, 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 ψ, given a label $\ell \in \mathbf{L}$ and address $a \in \mathbf{A}$, $\mathsf{Program}(\ell, a)$ modifies the DHT so that $\psi(\ell) = a$.

For the special case of Chord, given a label ℓ and an address a, the allocation parameter H_2 can be changed by programming the random oracle H_2 to output a when it is queried on ℓ[1].

[1] This is also true for every DHT we are aware of [30, 21, 40, 18].

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 small.

Definition 4.2 (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 $\ell \in \mathbf{L}$ and for all $S \subseteq \mathbf{C}$ with $|S| = \theta$,*

$$\Pr\Big[S \cap \mathsf{route}_\omega\Big(\mathsf{fe}_\phi(\ell), \mathsf{server}_{\omega,\psi}(\ell)\Big) \neq \emptyset)\Big] \leq \varepsilon$$

where the probability is over the coins of Alloc *and* FrontEnd *and where ε can depend on θ and $|\mathbf{C}|$.*

We will later see that the better the balance, the better the security gurantee of our EDDX scheme. The reason is that, inuitively, if an adversary sees a label with low probability, then it learns information about it with low probability.

Definition 4.3 (Balanced DHT)**.** *We say that a distributed hash table* $\mathsf{DHT} = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{FrontEnd}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$ *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* Overlay *and where ε and δ can depend on $\mathbf{C}$ and θ.*

Balance of Chord. We now analyze the balance of Chord. We show that with high probability, Chord samples an H_1 (i.e., a configuration χ) such that the visibility of any θ nodes is not too large. Showing this is non-trivial and requires us to bound the total lengths of θ (possibly non-contiguous) arcs in χ. Let $\mathsf{sumarcs}(\chi_{\mathbf{C}}, x)$ be the random variable denoting the total lengths of *any* x arcs in configuration χ.

One way to bound $\mathsf{sumarcs}(\chi_{\mathbf{C}}, x)$ is to bound the length of the largest arc, and then use a union bound on it for x arcs. Precisely, if the length of the largest arc is at most λ, then the total length of any θ arcs can be at most $\theta\lambda$. Unfortunately, this is a very weak bound, and we improve it by noticing that there cannot be a lot of very large arcs in a configuration. Intuitively, if one arc is on the larger side, then others will be on the smaller side. Formally speaking, the arc lengths are negatively dependent on each other, and we use the following result from [7]. We adapt their theorem in our notation.

Theorem 4.4 ([7])**.** *Let $\chi_{\mathbf{C}}$ be a configuration chosen uniformly at random. Then for $\theta \leq 4ne^{-2}$,*

$$\Pr\Big[\mathsf{sumarcs}(\chi_{\mathbf{C}}, \theta) \leq \frac{6\theta|\mathbf{A}|}{n}\log\Big(\frac{n}{\theta}\Big)\Big] \geq 1 - o\Big(\frac{1}{n}\Big)$$

Notice that this bound is only $O(\log \frac{n}{\theta})$ away from the optimal: in the best case, all the arcs are of average length $|\mathbf{A}|/n$, setting a lower bound of $\theta|\mathbf{A}|/n$ on $\mathsf{sumarcs}(\chi_{\mathbf{C}}, \theta)$.

Also notice that this theorem in some sense bounds the probability that a label will be stored at one of the θ adversarial nodes. This is because, Chord maps labels to addresses with uniform probability, and Theorem 4.4 bounds the fraction of the address space that the adversary holds in its arcs. Formally, for a set of nodes I such that $|I| = \theta < 4ne^{-2}$, with probability at least $1 - o(\frac{1}{n})$,

$$\begin{aligned}
\Pr\Big[\mathsf{server}(\ell) \in I\Big] &= \Pr\Big[H_2(\ell) \in \bigcup_{N \in I} \mathsf{arc}(N)\Big] \\
&= \frac{|\bigcup_{N \in I} \mathsf{arc}_{\chi}(N)|}{|\mathbf{A}|} \\
&\leq \frac{\mathsf{sumarcs}(\chi, |I|)}{|\mathbf{A}|} \\
&\leq \frac{6\theta}{n} \log\left(\frac{n}{\theta}\right) \qquad (1)
\end{aligned}$$

Finally, we bound the probability that a label is routed by an adversarial node. The main idea is the following: for all labels ℓ, given a random front end node (due to H_3), and an (almost) random destination node (due to H_2), the adversary cannot place itself on ℓ's route with very high probability, especially when Chord ensures that routes are at most $\log n$ long. Formally, we show the following:

Theorem 4.5. *Let $\chi_{\mathbf{C}}$ be a configuration chosen uniformly at random. Then for all labels ℓ, and for all $I \subseteq \mathbf{C} \setminus \{\mathsf{fe}(\ell), \mathsf{server}(\ell)\}$, with $|I| = \theta \leq n/\log n$,*

$$\Pr\Big[I \cap \mathsf{route}(\mathsf{fe}(\ell), \mathsf{server}(\ell)) \neq \emptyset\Big] \leq \frac{\theta \log n}{n}. \qquad (2)$$

Proof. For all $\ell \in \mathbf{L}$, let $\mathcal{E}$ be the event that at least one of the nodes in I is on the path to the server of ℓ. Precisely,

$$\mathcal{E} = \{I \cap \mathsf{route}(\mathsf{fe}(\ell), \mathsf{server}(\ell)) \neq \emptyset\}$$

By the union bound and the law of total probability, we have that,

$$\begin{aligned}
\Pr[\,\mathcal{E}\,] &= \Pr[\,I \cap \mathsf{route}(\mathsf{fe}(\ell), \mathsf{server}(\ell)) \neq \emptyset\,] \\
&\leq \sum_{N \in I} \Pr[\,N \in \mathsf{route}(\mathsf{fe}(\ell), \mathsf{server}(\ell))\,] \\
&= \sum_{N \in I} \sum_{N_s \in \mathbf{C}} \sum_{N_d \in \mathbf{C}} \Pr[\,N \in \mathsf{route}(N_s, N_d) \mid \mathsf{server}(\ell) = N_d \wedge \mathsf{fe}(\ell) = N_s\,] \cdot \\
&\qquad\qquad \Pr[\,\mathsf{fe}(\ell) = N_s \mid \mathsf{server}(\ell) = N_d\,] \cdot \Pr[\,\mathsf{server}(\ell) = N_d\,] \\
&= \sum_{N \in I} \sum_{N_s \in \mathbf{C}} \sum_{N_d \in \mathbf{C}} \Pr[\,N \in \mathsf{route}(N_s, N_d) \mid \mathsf{server}(\ell) = N_d \wedge \mathsf{fe}(\ell) = N_s\,] \cdot \\
&\qquad\qquad \Pr[\,\mathsf{fe}(\ell) = N_s\,] \cdot \Pr[\,\mathsf{server}(\ell) = N_d\,] \qquad (3)
\end{aligned}$$

where, the last equality follows from the fact that $\mathsf{fe}(\ell)$ and $\mathsf{server}(\ell)$ are chosen independently. But note that,

$$\Pr[\,N \in \mathsf{route}(N_s, N_d) \mid \mid \mathsf{server}(\ell) = N_d \wedge \mathsf{fe}(\ell) = N_s\,] \leq \frac{\log n}{n}$$

which follows from the fact that path lengths in Chord are at most $\log n$. Substituting this in Eq.

(3) we get,

$$\begin{aligned}
\Pr[\mathcal{E}] &\leq \sum_{N\in I}\sum_{N_s\in\mathbf{C}}\sum_{N_d\in\mathbf{C}}\frac{\log n}{n}\cdot\Pr[\,\mathsf{fe}(\ell)=N_s\,]\cdot\Pr[\,\mathsf{server}(\ell)=N_d\,]\\
&\leq \sum_{N\in I}\frac{\log n}{n}\sum_{N_s\in\mathbf{C}}\sum_{N_d\in\mathbf{C}}\Pr[\,\mathsf{fe}(\ell)=N_s\,]\cdot\Pr[\,\mathsf{server}(\ell)=N_d\,]\\
&= \sum_{N\in I}\frac{\log n}{n}\\
&= \frac{\theta\log n}{n}
\end{aligned}$$

□

We now use Equations 1 and 2 to show that Chord is balanced.

Theorem 4.6. *Chord, is* $(\varepsilon,\delta,\theta)$ *balanced for:*

$$\varepsilon=\frac{8\theta\log n}{n}\quad\text{and}\quad\theta\leq\frac{n}{8\log n}\quad\text{and}\quad\delta=1-o\left(\frac{1}{n}\right)$$

Proof. Given a χ, we bound the probability that for a label ℓ, an adversarial node is on the route from $\mathsf{fe}(\ell)$ to $\mathsf{server}(\ell)$. There are three ways in which this can happen: (1) front end node is corrupted, or (2) the storage node is corrupted, or (3) one of the nodes on the route (excluding ($\mathsf{fe}(\ell)$) and $\mathsf{server}(\ell)$) are corrupted. Therefore, we have that:

$$\begin{aligned}
&\Pr\Big[I\cap\mathsf{route}(\mathsf{fe}(\ell),\mathsf{server}(\ell))\neq\emptyset\Big]\\
&\quad=\Pr\Big[\mathsf{fe}(\ell)\in I\Big]+\Pr\Big[\mathsf{server}(\ell)\in I\Big]+\Pr\Big[I\cap\mathsf{route}(\mathsf{fe}(\ell),\mathsf{server}(\ell))\neq\emptyset\;\Big|\;\mathsf{fe}(\ell)\notin I\wedge\mathsf{server}(\ell)\notin I\Big]\\
&\quad=\Pr\Big[H_3(\ell)\in I\Big]+\frac{6\theta}{n}\log\left(\frac{n}{\theta}\right)+\Pr\Big[I\cap\mathsf{route}(H_3(\ell),\mathsf{server}(\ell))\neq\emptyset\;\Big|\;H_3(\ell)\notin I\wedge\mathsf{server}(\ell)\notin I\Big]\\
&\quad\leq\frac{\theta}{n}+\frac{6\theta}{n}\log\left(\frac{n}{\theta}\right)+\frac{\theta\log n}{n}\\
&\quad\leq\frac{8\theta\log n}{n} \qquad (4)
\end{aligned}$$

where the second and third inequalities follow from Equations 1 and 2. Note that the bound above only makes sense for $\theta<n/8\log n$. Finally, we know that Equation 1 holds with probability $1-o(\frac{1}{n})$, therefore, we conclude that Chord is balanced for the given values of ε, δ and θ. □

Note that assigning labels uniformly at random to nodes would achieve $\varepsilon=\theta/n$, whereas Theorem 4.6 achieves $\varepsilon=O(\theta\log n/n)$. This shows that the balance of Chord is only $\log n$ factor away from optimal balance which is very good given that the optimal balance is achieved with no routing at all.

5 A DDX Encryption Scheme in the Perpetual Setting

In this section, we describe an EDDX scheme BDX in the perpetual setting where nodes are fixed throughout the lifetime of the EDDX. BDX relies on simple cryptographic primitives and a non-committing and balanced DHT.

Let $\mathsf{DHT} = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{FrontEnd}, \mathsf{Put}, \mathsf{Get})$ 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 dictionary scheme $\mathsf{EDDX} = (\mathsf{Init}, \mathsf{Put}, \mathsf{Get})$ that works as follows:

```
• $\mathsf{Init}_{\mathcal{C},\mathbf{C}}(1^k; \perp_1; \ldots; \perp_n)$:
    1. $\mathcal{C}$ samples $K_1 \xleftarrow{\$} \{0,1\}^k$ and compute $K_2 \leftarrow \mathsf{SKE.Gen}(1^k)$;
    2. $\mathcal{C}$ computes $\omega \leftarrow \mathsf{DHT.Overlay}(n)$, $\psi \leftarrow \mathsf{DHT.Alloc}(n)$, and $\phi \leftarrow \mathsf{DHT.FrontEnd}(n)$;
    3. $\mathcal{C}$ sends $\Gamma = (\omega, \psi, \phi)$ to all the active nodes;
    4. $\mathcal{C}$ outputs $K = (K_1, K_2)$;
    5. all the nodes $N_i \in \mathbf{C}$ execute $(st_i, \mathsf{DX}_i) \leftarrow \mathsf{DHT.Init}(\Gamma)$;
    6. all the nodes $N_i \in \mathbf{C}$ output $\mathsf{EDDX}_i = (st_i, \mathsf{DX}_i)$;
• $\mathsf{Put}_{\mathcal{C},\mathbf{C}}(K, \ell, v; \mathsf{EDDX}_1; \ldots; \mathsf{EDDX}_n)$ :
    1. $\mathcal{C}$ parses $K$ as $(K_1, K_2)$;
    2. $\mathcal{C}$ compute $t := F_{K_1}(\ell)$;
    3. $\mathcal{C}$ compute $e \leftarrow \mathsf{SKE.Enc}(K_2, v)$;
    4. all the nodes $N_i \in \mathbf{C}$ parse $\mathsf{EDDX}_i$ as $(st_i, \mathsf{DX}_i)$;
    5. $\mathcal{C}$ and the nodes $N_i \in \mathbf{C}$ execute
           $(\perp; (st_1, \mathsf{DX}'_1); \ldots; (st_n, \mathsf{DX}'_n)) \leftarrow \mathsf{DHT.Put}(t, e; (st_1, \mathsf{DX}_1); \ldots; (st_n, \mathsf{DX}_n))$;
    6. all the nodes $N_i \in \mathbf{C}$ output $\mathsf{EDDX}'_i = (st_i, \mathsf{DX}'_i)$;
• $\mathsf{Get}_{\mathcal{C},\mathbf{C}}(K, \ell; \mathsf{EDDX}_1; \ldots; \mathsf{EDDX}_n)$:
    1. $\mathcal{C}$ parses $K$ as $(K_1, K_2)$
    2. $\mathcal{C}$ computes $t := F_{K_1}(\ell)$
    3. all the nodes $N_i \in \mathbf{C}$ parse $\mathsf{EDDX}_i$ as $(st_i, \mathsf{DX}_i)$;
    4. $\mathcal{C}$ and the nodes $N_i \in \mathbf{C}$ execute
           $(e; \perp_1; \ldots; \perp_n) \leftarrow \mathsf{DHT.Get}(t\ ; (st_1, \mathsf{DX}_1); \ldots; (st_n, \mathsf{DX}_n))$;
    5. if $e \neq \perp$, client $\mathcal{C}$ computes and outputs $v \leftarrow \mathsf{SKE.Dec}(K_2, e)$;
```

Figure 3: BDX: A DDX encryption scheme in the perpetual setting.

Overview. The scheme $\mathsf{BDX} = (\mathsf{Init}, \mathsf{Put}, \mathsf{Get})$ is described in detail in Figure 3 and, at a high level, works as follows. It makes black-box use of a distributed hash table $\mathsf{DHT} = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{FrontEnd}, \mathsf{InitPut}, \mathsf{Get})$, a pseudo-random function F and a symmetric-key encryption scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. At the time of Init, the client uses a security parameter 1^k 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 executes $\mathsf{DHT.Overlay}$, $\mathsf{DHT.Alloc}$, and $\mathsf{DHT.FrontEnd}$ to generate and forward the DHT paramters ω, ψ and ϕ to all the active nodes in $\mathbf{C}$. The nodes then input the DHT parameters into $\mathsf{DHT.Init}$ to setup their local dictionaries and states. To store a label/value pair (ℓ, v), the client first computes $t := F_{K_1}(\ell)$ and $e \leftarrow \mathsf{Enc}(K_2, v)$ and then executes $\mathsf{DHT.Put}(t, e)$ with the nodes. To retrieve the value associated with a label ℓ, the client computes $t := F_{K_1}(\ell)$ and executes $e \leftarrow \mathsf{DHT.Get}(t)$ with the nodes. It then decrypts $\mathsf{SKE.Dec}(K, e)$ the value returned and outputs it.

Security. We now describe the leakage of EDDX. 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 $(\mathtt{put}, \mathsf{opeq}(\ell))$
     (b) else if $\mathsf{op} = \mathtt{get}$, output $(\mathtt{get}, \mathsf{opeq}(\ell))$
  3. else if $b_\ell = 0$
     (a) output $\perp$

where opeq is the *operation 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 [13, 11]. 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; (2) start at a corrupted front end node; or (3) 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 DHT parameters $\Gamma = (\omega, \psi, \phi)$, 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, ψ and ϕ 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 our EDDX scheme is $\mathcal{L}_\varepsilon$-secure with probability that is negligibly close to $1 - \delta$ when its underlying DHT is $(\varepsilon, \delta, \theta)$-balanced.

Theorem 5.1. *If* $|I| \leq \theta$ *and if* DHT *is* $(\varepsilon, \delta, \theta)$*-balanced and has non-committing allocation, then* BDX *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 $\omega \leftarrow \mathsf{DHT.Overlay}(n)$, $\phi \leftarrow \mathsf{DHT.FrontEnd}(n)$, 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 $K \leftarrow \mathsf{SKE.Gen}(1^k)$. When a put/get operation is executed, Sim receives from F_{DHT} the leakage

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

If $\lambda = \perp$ then Sim does nothing. If $\lambda \neq \perp$, then Sim checks the operation equality to see if the label has been seen in the past. There are two cases:

1. **Label was not seen in the past:** If the label was not seen in the past (as deduced from operation equality), it sets $t \overset{\$}{\leftarrow} \{0,1\}^d$, and samples and stores a bit

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

   where, $p' \overset{def}{=} \Pr\left[\, \psi(t) \in \mathsf{Vis}(\mathsf{fe}_\phi(t), I) \,\right]$. Note that, this is indeed a valid Bernoulli distribution since

   $$p' = \Pr\left[\, \psi(t) \in \mathsf{Vis}(\mathsf{fe}(t), I) \,\right] \leq \varepsilon,$$

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

   If $b' = 0$, it does nothing, but if $b' = 1$, it computes $e \leftarrow \mathsf{SKE.Enc}(K, 0)$, sets the frontend node $N_s \leftarrow \mathsf{fe}(t)$, samples an address $a \leftarrow \Delta_1(\mathsf{Vis}(\mathsf{fe}(t), I))$. It then programs ψ to map t to a. Finally, if the operation was a put, it executes $\mathsf{DHT.Put}(t, e)$, otherwise it executes $\mathsf{DHT.Get}(t)$.

2. **Label was seen in the past:** If the label was seen in the past (as deduced from operation equality), Sim retrieves the bit b' that was previously sampled. If $b' = 0$, then it does nothing, but if $b' = 1$ it sets t to the d-bit value previously used, and computes $e \leftarrow \mathsf{SKE.Enc}(K, 0)$. 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 $\mathsf{Sim}(\mathtt{put}, \mathsf{opeq}(\ell))$, and if $b_\ell = 1$ and $\mathsf{op} = (\mathtt{get}, \ell)$, we replace the Get operation with $\mathsf{Sim}(\mathtt{get}, \mathsf{opeq}(\ell))$. 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 $\psi(t_i) \in \mathsf{Vis}(\mathsf{fe}(t_i), I)$. 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

$$\begin{aligned}
&\Pr\left[\,\mathbf{view_3}^i(I) = \mathbf{v}\,\right] \\
&\quad = \Pr\left[\,\mathbf{view_3}^i(I) = \mathbf{v} \wedge \mathcal{Q}_i\,\right] + \Pr\left[\,\mathbf{view_3}^i(I) = \mathbf{v} \wedge \overline{\mathcal{Q}_i}\,\right] \\
&\quad = \Pr\left[\,\mathbf{view_3}^i(I) = \mathbf{v} \mid \mathcal{Q}_i\,\right] \cdot \Pr\left[\,\mathcal{Q}_i\,\right] + \Pr\left[\,\mathbf{view_2}^i(I) = \mathbf{v} \mid \overline{\mathcal{Q}_i}\,\right] \cdot \Big(1 - \Pr\left[\,\mathcal{Q}_i\,\right]\Big) \\
&\quad = \Pr\left[\,\mathbf{view_3}^i(I) = \mathbf{v} \mid \mathcal{Q}_i\,\right] \cdot \Pr\left[\,\mathcal{Q}_i\,\right] \qquad (5)
\end{aligned}$$

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

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

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

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

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 allocation ψ generated using Alloc, whereas in the latter, the labels t_i are programmed to an address a sampled from $\Delta_1(\mathsf{Vis}(\mathsf{fe}(t_i), I))$. We show, however, that in both cases, the labels t_i are allocated with the same probability distribution. In Game_2, for all $a \in \mathbf{A}$, we have,

$$\begin{aligned}\Pr\left[\,\psi(t_i) = a \mid \mathcal{E}_i\,\right] &= \frac{\Pr\left[\,\psi(t_i) = a \wedge \mathcal{E}_i\,\right]}{\Pr\left[\,\mathcal{E}_i\,\right]} \\ &= \frac{\Pr\left[\,\psi(t_i) = a \wedge \psi(t_i) \in \mathsf{Vis}(\mathsf{fe}(t_i), I)\,\right]}{\Pr\left[\,\psi(t_i) \in \mathsf{Vis}(\mathsf{fe}(t_i), I)\,\right]}\end{aligned} \tag{6}$$

We see that for $a \notin \mathsf{Vis}(\mathsf{fe}(t_i), I)$, the nuemrator is 0, and therefore for $a \notin \mathsf{Vis}(\mathsf{fe}(t_i), I)$, we have that,

$$\Pr\left[\,\psi(t_i) = a \mid \mathcal{E}_i\,\right] = 0$$

However, for $a \in \mathsf{Vis}(\mathsf{fe}(t_i), I)$, Eqn 6 evaluates to:

$$\Pr\left[\,\psi(t_i) = a \mid \mathcal{E}_i\,\right] = \frac{\Pr\left[\,\psi(t_i) = a\,\right]}{\Pr\left[\,\psi(t_i) \in \mathsf{Vis}(\mathsf{fe}(t_i), I)\,\right]}$$

which follows from the fact that for $a \in \mathsf{Vis}(\mathsf{fe}(t_i), I)$, the event $\{\psi(t_i) = a\} \subseteq \mathcal{E}_i$.
In Game_3, since the simulator only programs t_i to an address in $\mathsf{Vis}(\mathsf{fe}(t_i), I)$ when $b_i = 1$ and $b'_i = 1$, we have that for all $a \notin \mathsf{Vis}(\mathsf{fe}(t_i), I)$,

$$\Pr\left[\,\psi(t_i) = a \mid \mathcal{Q}_i\,\right] = 0$$

However, for $a \in \mathsf{Vis}(\mathsf{fe}(t_i), I)$, we have that,

$$\Pr\left[\,\psi(t_i) = a \mid \mathcal{Q}_i\,\right] = \frac{\Pr\left[\,\psi(t_i) = a\,\right]}{\Pr\left[\,\psi(t_i) \in \mathsf{Vis}(\mathsf{fe}(t_i), I)\,\right]}$$

where the first equation follows from the fact that a is sampled from $\Delta_1(\mathsf{Vis}(\mathsf{fe}(t_i), I))$. Since, for all i, conditioned on $\mathcal{Q}_i$ and $\mathcal{E}_i$, labels are allocated to addresses with the same distribution in both games and since this is the only difference between the games,

$$\Pr\left[\mathbf{view_3}^i(I) = \mathbf{v} \mid \mathcal{Q}_i\right] = \Pr\left[\mathbf{view_2}^i(I) = \mathbf{v} \mid \mathcal{E}_i\right]. \tag{7}$$

Plugging Eq. 7 into Eq. 5, we have that for all i and all $\mathbf{v}$,

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

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 as the last time it saw t_i. □

Security of the Chord-based BDX. In the following Corollary we formally state the security of BDX scheme when its underlying DHT is instantiated with Chord. The proof follows directly from the fact that Chord has non-committing allocations and that it is balanced for:

$$\varepsilon = \frac{8\theta \log n}{n} \quad \text{and} \quad \theta \leq \frac{n}{8 \log n} \quad \text{and} \quad \delta = 1 - o\left(\frac{1}{n}\right)$$

Corollary 5.2. *If $|\mathbf{L}| = \Theta(2^k)$, $|I| = \theta \leq n/\log^2 n$, and if* EDDX *is instantiated with Chord, then it is $\mathcal{L}_\varepsilon$-secure with probability at least $1 - o(\frac{1}{n}) - \mathsf{negl}(k)$ in the random oracle model, where $\varepsilon = \frac{8\theta \log n}{n}$.*

Proof. The corollary follows from Theorem 5.1, Theorem 4.6 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. □

Notice that if the number of corruptions θ is at most $n/(\alpha \log n)$, where $\alpha > 8$, then we get that $\varepsilon = O(1/\alpha)$. 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 $n/\log n$ nodes are corrupted then, the adversary can expect to learn the operation equality of an $O(1/\alpha)$ fraction of client queries. Note that this confirms the intuition that distributing an STE scheme suppresses its leakage.

Efficiency of BDX. Our construction BDX does not add any asymptotic overhead to time, round, communication and storage complexities of the underlying DHT.

6 Transient DHTs

In this section, we consider DHTs in the transient setting. Transient DHTs are the same as perpetual DHTs with the difference that nodes are not known a-priori and can join and leave at any time. Instead of using perpetual DHTs as a building block, we will instead use transient DHTs in our construction. Similar to the perpetual setting, we will use Chord as our running example.

Syntax. Transient DHTs are a collection of eight protocols $\mathsf{DHT}^+ = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{FrontEnd}, \mathsf{Init}, \mathsf{Put}, \mathsf{Get}, \mathsf{Leave}, \mathsf{Join})$. The first six algorithms are same as in the perpetual setting. The seventh is a protocol Leave executed between a node $N \in \mathbf{C}$ when it wishes to leave the network and all the other nodes in the network. Leave takes as input the states and dictionaries of all the nodes in the network, and outputs an updated state and an updated dictionary at the remaining nodes. The eighth is also a protocol Join is the same as the Leave protocol with the difference that it is executed between a node $N \in \mathbf{N} \setminus \mathbf{C}$ that wishes to join the network and the nodes already in the network. 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.

Leaves and joins in Chord. The Chord paper [40] does not precisely specify how joins and leaves should be handled. In particular, when a node N leaves, it describes which nodes should receive N's pairs, but it does not describe exactly how the pairs should get there. Similarly, when a node N joins, it describes which pairs should move to N, but it does not describe how these pairs

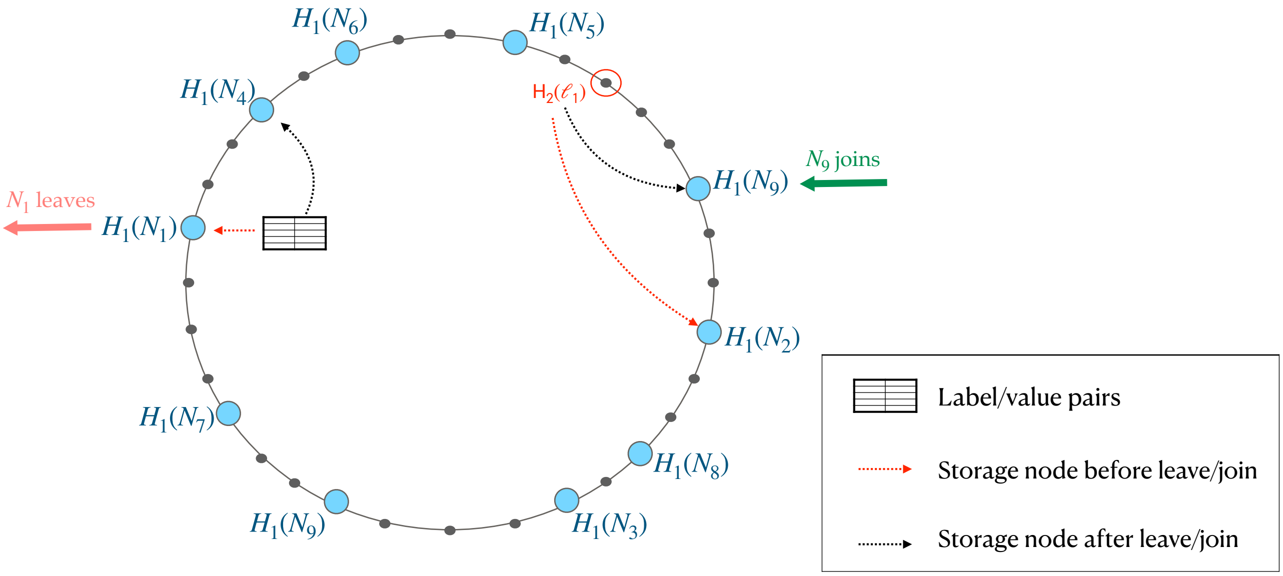

Figure 4: Leaves and joins in Chord. When a node joins, label/value pairs stored at its successor are rehashed, and as a result some of the pairs stored at the successor move to the newly joined node. For example, when N_9 joins, ℓ_1 moves to N_9 while ℓ_2 continues to be stored at N_2. When a node leaves, all its pairs are moved to its successor. For example, when N_1 leaves, all its pairs are moved to N_4.

should move to N. Because of this, we describe 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.

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 $\mathsf{succ}_{\chi_\mathbf{C}}(H_1(N))$.

Intuitively, when a node N joins, it splits the arc of its successor, and all the pairs that hashed to the part of the arc that is now the arc of N, are moved to N. As shown in Figure 4, when a new node N_9 joins, ℓ_1 which was previosuly stored at N_2 moves to N_9. Similarly, when a node N leaves, its arc becomes a part of its successor N''s arc. Hence all the pairs that initially hashed to N's arc, now hash to N''s arc, and hence move to N'. For example, in Figure 4, when N_1 leaves, all its pairs move to N_4.

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

Definition 6.1 (Stability)**.** *We say that a transient distributed hash table* $\mathsf{DHT}^+ = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{FrontEnd}, \mathsf{Init}, \mathsf{Put}, \mathsf{Get}, \mathsf{Leave}, \mathsf{Join})$ *is* $(\varepsilon, \delta, \theta)$*-stable if*

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

where the probability is over the choice of ω*, and* ε *and* θ *are functions of* $\mathbf{C}$*.*

Stability of Chord. Recall that in the perpetual setting, we have a single configuration $\chi_{\mathbf{C}}$ corresponding to a fixed set of active nodes $\mathbf{C}$. However, in the transient setting, we have multiple configurations — every time a node leaves/joins, the set $\mathbf{C}$ changes and hence the configuration $\chi_{\mathbf{C}}$ changes. In order to show that Chord is stable, we need to show that all the configurations $\chi_{\mathbf{C}}$ are balanced. And in order to do so, for each configuration, we need to bound the probability of (1) the front end node being corrupted, (2) a label being stored at a corrupted node, and (3) a label being routed by a corrupted node.

Notice that given a configuration $\chi_{\mathbf{C}}$, the probability that a front-end node is corrupted only depends on the fraction of nodes currently corrupted and hence is always $\theta/|\mathbf{C}|$. Similarly, the probability that a label is routed by a corrupted node only depends on the path lengths, which are $\log|\mathbf{C}|$ long in any configuration. Therefore, we bound the probability of (3) by $\theta\log|\mathbf{C}|/|\mathbf{C}|$ following the same argument we used to bound the probability of a label being routed by a corrupted node in the persistent setting (Theorem 4.5).

Notice that bounding (2) simultaneously for all the configurations is non-trivial, and it is because the event of a label being stored at a corrupted node is not independent across configurations. For example, if an uncorrupted node adjacent to a corrupted node leaves, in the new configuration, the corrupted node also occupies the arc of the node that left the system, and hence it has a higher likelihood of storing a label than what it previously had in the old configuration.

As before, for a given configuration $\chi_{\mathbf{C}}$, in order to bound (2), we upper bound the total lengths of the θ largest arcs in $\chi_{\mathbf{C}}$. However, instead of bounding it as a function of the number of currently active nodes $|\mathbf{C}|$, we bound it as function of the size of the namepspace $|\mathbf{N}|$. We rely 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 arc becomes visible to some other (currently active) node.

But how can we use these observations to bound the total lengths of θ largest arcs in $\chi_{\mathbf{C}}$ using the bound 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 arc 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 $\mathsf{sumarcs}(\chi_{\mathbf{C}},\theta) \leq \mathsf{sumarcs}(\chi_{\mathbf{N}},\theta+|\mathbf{N}|-|\mathbf{C}|)$. We next show that for large enough $\mathbf{C}$'s, i.e., when $|\mathbf{C}|\geq|\mathbf{N}|-d$, $\mathsf{sumarcs}(\chi_{\mathbf{N}},\theta+|\mathbf{N}|-|\mathbf{C}|) \leq \mathsf{sumarcs}(\chi_{\mathbf{N}},\theta+d)$. Finally, we bound $\mathsf{sumarcs}(\chi_{\mathbf{N}},\theta+d)$ using the same result from [7] which we used earlier.

Theorem 6.2. *Let* $\chi_{\mathbf{N}}$ *be a configuration drawn u.a.r and let* d *be a positive integer such that* $\theta+d\leq 4|\mathbf{N}|e^{-2}$*. Then,*

$$\Pr\bigg[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq|\mathbf{N}|-d}\Big\{\mathsf{sumarcs}(\chi_{\mathbf{C}},\theta) \leq \frac{6(\theta+d)|\mathbf{A}|}{|\mathbf{N}|}\log\Big(\frac{|\mathbf{N}|}{\theta+d}\Big)\Big\}\bigg] \geq 1-o\Big(\frac{1}{|\mathbf{N}|}\Big)$$

Proof. Proof follows directly from the observation that for all the configurations $\chi_{\mathbf{C}}$, the probabibilty of the event $\{\mathsf{sumarcs}(\chi_{\mathbf{C}}, \theta) < \alpha\}$ is at least as the probability of the event $\{\mathsf{sumarcs}(\chi_{\mathbf{N}}, \theta + d) < \alpha\}$. Precisely,

$$\begin{aligned}\Pr\Bigg[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq|\mathbf{N}|-d}\Bigg\{\mathsf{sumarcs}(\chi_{\mathbf{C}}, \theta) \leq \frac{6(\theta+d)|\mathbf{A}|}{|\mathbf{N}|}\log\Bigg(\frac{|\mathbf{N}|}{\theta+d}\Bigg)\Bigg\}\Bigg]\\ \geq \Pr\Bigg[\mathsf{sumarcs}(\chi_{\mathbf{N}}, \theta+d) \leq \frac{6(\theta+d)|\mathbf{A}|}{|\mathbf{N}|}\log\Bigg(\frac{|\mathbf{N}|}{\theta+d}\Bigg)\Bigg]\\ \geq 1 - o\Bigg(\frac{1}{|\mathbf{N}|}\Bigg)\end{aligned}$$

where the last step follows from Theorem 4.4. □

We finally show that Chord is $(\varepsilon, \delta, \theta)$ stable, where ε and θ depend on the number of nodes currently active.

Theorem 6.3. *Transient Chord is $(\varepsilon, \delta, \theta)$ stable for*

$$\varepsilon = \frac{6(\theta+d)\log|\mathbf{N}|}{|\mathbf{N}|} + \frac{2\theta\log|\mathbf{C}|}{|\mathbf{C}|} \quad \text{and} \quad \theta \leq 4|\mathbf{N}|e^{-2} - d \quad \text{and} \quad \delta = 1 - o\Bigg(\frac{1}{|\mathbf{N}|}\Bigg),$$

where d is some positive integer.

Proof. For a given set of active nodes $\mathbf{C}$, let $\mathcal{E}_{\mathbf{C}}$ be the following event:

$$\Bigg\{\Pr\Big[I \cap \mathsf{route}_{\chi_{\mathbf{C}}}(\mathsf{fe}_{H_3}(\ell), \mathsf{server}_{\chi_{\mathbf{C}}}(\ell)) \neq \emptyset\Big] \leq \varepsilon_{\mathbf{C}}\Bigg\}$$

Let us rewrite $\varepsilon_{\mathbf{C}} = \varepsilon^1_{\mathbf{C}} + \varepsilon^2_{\mathbf{C}} + \varepsilon^3_{\mathbf{C}}$, where:

$$\begin{aligned}\varepsilon^1_{\mathbf{C}} &= \frac{\theta_{\mathbf{C}}}{|\mathbf{C}|}\\ \varepsilon^2_{\mathbf{C}} &= \frac{6(\theta+d)\log|\mathbf{N}|}{|\mathbf{N}|} + \frac{2\theta\log|\mathbf{C}|}{|\mathbf{C}|}\\ \varepsilon^3_{\mathbf{C}} &= \frac{\theta_{\mathbf{C}}\log|\mathbf{C}|}{|\mathbf{C}|}\end{aligned}$$

and let us define three new events as follows:

$$\begin{aligned}\mathcal{E}^1_{\mathbf{C}} &= \Bigg\{\Pr\Big[\mathsf{fe}_{H_3}(\ell) \in I\Big] \leq \varepsilon^1_{\mathbf{C}}\Bigg\}\\ \mathcal{E}^2_{\mathbf{C}} &= \Bigg\{\Pr\Big[\mathsf{server}_{\chi_{\mathbf{C}}}(\ell) \in I\Big] \leq \varepsilon^2_{\mathbf{C}}\Bigg\}\\ \mathcal{E}^3_{\mathbf{C}} &= \Bigg\{\Pr\Bigg[\Big\{I \cap \mathsf{route}_{\chi_{\mathbf{C}}}(\mathsf{fe}_{H_3}(\ell), \mathsf{server}_{\chi_{\mathbf{C}}}(\ell))\Big\} \neq \emptyset \;\Bigg|\; \mathsf{fe}_{H_3}(\ell) \notin I \wedge \mathsf{server}_{\chi_{\mathbf{C}}}(\ell) \notin I\Bigg] \leq \varepsilon^3_{\mathbf{C}}\Bigg\}\end{aligned}$$

Then,

$$\Pr\Bigg[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta}\ \bigwedge_{i\in[3]} \mathcal{E}^i_{\mathbf{C}}\Bigg] = \Pr\Bigg[\bigwedge_{i\in[3]}\ \bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta} \mathcal{E}^i_{\mathbf{C}}\Bigg] = \prod_{i\in[3]}\Pr\Bigg[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta} \mathcal{E}^i_{\mathbf{C}}\Bigg] \tag{8}$$

where the last inequality follows from the fact that $\mathcal{E}^i_{\mathbf{C}}$'s are independent. We next evaluate the three terms inside the product one by one.

We start by evaluating $\Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta} \mathcal{E}^1_{\mathbf{C}}\right]$. We notice that given any configuration, probability that front end of label is a corrupted node only depends on the fraction of nodes currently corrupted, and not on how nodes are arranged in a configuration. Precisely, for all $\mathbf{C}$,

$$\Pr\left[\mathsf{fe}_{H_3}(\ell) \in I\right] = \frac{\theta_{\mathbf{C}}}{|\mathbf{C}|}$$

Therefore, for all $\mathbf{C}$, the event $\mathcal{E}^1_{\mathbf{C}}$ always occurs, and hence

$$\Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta} \mathcal{E}^1_{\mathbf{C}}\right] = \Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta} 1\right] = 1 \tag{9}$$

Similarly, we notice that the probabibilty that a corrupted node falls on the path of a label to its server (excluding the ending points of the path) solely depends on the route lengths, which given any configuration, are always $\log|\mathbf{C}|$ long. Therefore, the event $\mathcal{E}^3_{\mathbf{C}}$ always occurs with probability 1, and hence,

$$\Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta} \mathcal{E}^3_{\mathbf{C}}\right] = 1 \tag{10}$$

However, the probability that a label is stored on a corrupted node is not independent across configurations, and hence the event $\mathcal{E}^2_{\mathbf{C}}$ is not independent for all $\mathbf{C}$. In particular,

$$\Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta} \mathcal{E}^2_{\mathbf{C}}\right] = \Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta} \left\{\mathsf{sumarcs}(\chi_{\mathbf{C}}, \theta) \leq \varepsilon^2_{\mathbf{C}}\right\}\right] \geq 1-\delta \tag{11}$$

where the last step follows from Theorem 6.2. From Eqnations 8–11, we conclude that:

$$\Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta} \;\bigwedge_{i\in[3]} \mathcal{E}^i_{\mathbf{C}}\right] \geq 1-\delta$$

Notice that $\mathcal{E}^1_{\mathbf{C}} \wedge \mathcal{E}^2_{\mathbf{C}} \wedge \mathcal{E}^3_{\mathbf{C}}$ imply the event $\mathcal{E}_{\mathbf{C}}$. And hence,

$$\Pr\left[\bigwedge_{\mathbf{C}\subseteq\mathbf{N}:|\mathbf{C}|\geq\theta} \mathcal{E}_{\mathbf{C}}\right] \geq 1-\delta$$

□

7 A DDX Encryption Scheme in the Transient Setting

In the transient setting, the scheme consists of $\mathsf{BDX}^+ = (\mathsf{Init}, \mathsf{Put}, \mathsf{Get}, \mathsf{AddNode}, \mathsf{RemoveNode})$ five protocols. The first three are exactly the same as the BDX scheme in the perpetual setting (See Figure 3). To remove a node N from the network, the client sends a message to node N indicating that it should leave the network. The node N then executes $\mathsf{DHT.Leave}$ which in turn moves it pairs to the remaining nodes in the network. Similarly, to add a node N, the client sends N a message, which in turn executes $\mathsf{DHT.Join}$. We start with a description of the leakage for add and remove node operations and then discuss the leakage for put and get operations.

Add and remove node 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. Now, when an add or remove node operation occurs, label/value pairs are moved throughout the network (e.g., during a remove node, 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 add/remove node 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 add/remove node 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). We denote the number of previously unseen pairs by κ.

The leakage profile. We are now ready to formally describe the leakage profile achieved by our construction BDX^+ in the transient setting.

- $\mathcal{L}_\varepsilon\left(\mathsf{DX}, \left\{(\mathsf{op}, \ell, v), (\mathsf{op}, N)\right\}\right)$:
    1. if $\mathsf{op} = \mathtt{get} \vee \mathtt{put}$ and ℓ has never been seen
        (a) sample and store $b_\ell \leftarrow \mathsf{Ber}(\varepsilon_{\mathbf{C}})$
    2. if $b_\ell = 1$
        (a) if $\mathsf{op} = \mathtt{put}$ output $(\mathtt{put}, \mathsf{opeq}(\ell))$
        (b) else if $\mathsf{op} = \mathtt{get}$ output $(\mathtt{get}, \mathsf{opeq}(\ell))$
    3. else if $b_\ell = 0$
        (a) Increment κ if $\mathsf{op} = \mathtt{put}$ and ℓ has never been seen
        (b) output $\perp$
    4. if $\mathsf{op} = \mathtt{removenode} \vee \mathtt{addnode}$
        (a) output (op, N, κ)
        (b) if $\mathsf{op} = \mathtt{removenode}$, $\mathbf{C} = \mathbf{C} \setminus N$
        (c) if $\mathsf{op} = \mathtt{addnode}$, $\mathbf{C} = \mathbf{C} \cup N$
        (d) set $b_\ell = 1$ for all the put labels that have been seen in the past
        (e) reset κ to 0

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

Theorem 7.1. *If $|I| \leq \theta$ and DHT^+ is $(\varepsilon, \delta, \theta)$-stable and has non-committing allocations, then BDX^+ 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 computes $\omega \leftarrow \mathsf{DHT}^+.\mathsf{Overlay}(n)$, $\phi \leftarrow \mathsf{DHT}^+.\mathsf{FrontEnd}(n)$, 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 $K \leftarrow \mathsf{SKE.Gen}(1^k)$. It then sets $I' = \mathbf{C} \cap I$.

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

$$\lambda \in \left\{ \left(\mathtt{removenode}, N, \kappa \right), \left(\mathtt{addnode}, N, \kappa \right) \right\}.$$

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 $a \leftarrow \Delta_1(\mathbf{A} \setminus \mathsf{Vis}(\mathsf{fe}(t_j), I'))$, programs ψ to map t_j to a, computes $N' \leftarrow \mathsf{server}(t_j)$, and adds (t_j, e_j) to $\mathsf{MM}[N']$. It also sets $b'_i = 1$ for all the labels it has received until now.

If the operation is a leave operation, it updates $\mathbf{C} = \mathbf{C} \setminus \{N\}$, updates the routing tables to exclude N, and executes $\mathsf{DHT.Put}(t, e)$ on all the (t, e) pairs stored in $\mathsf{MM}[N]$, updating MM according to how pairs move. It finally, resets $\mathsf{MM}[N]$ to $\perp$, and recomputes $I' = I \cap \mathbf{C}$.

If the operation is a join operation, it updates $\mathbf{C} = \mathbf{C} \cup \{N\}$, updates the routing tables to include N, and executes $\mathsf{DHT.Put}(t, e)$ on all the (t, e) pairs stored in MM for all the nodes, updating MM according to how pairs move. Finally, it recomputes $I' = I \cap \mathbf{C}$.

When a put/get operation is executed, it does the exact same thing as in the case of persistent setting. There are only two differences: (1) for a put operation if $b'_i = 0$, the simulator generates a random (t, e) pair, and program $\psi(t) = a$, where $a \leftarrow \Delta_1(\mathbf{A} \setminus \mathsf{Vis}(\mathsf{fe}(t_j), I'))$, and (2) it updates MM in the process of executing Get and Put as well.

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 $\mathsf{Sim}(\mathtt{put}, \mathsf{opeq}(\ell))$ and if $\mathsf{op} = (\mathtt{get}, \ell)$ we replace the Get operation with $\mathsf{Sim}(\mathtt{get}, \mathsf{opeq}(\ell))$. If $b_\ell = 0$, we do nothing. If however $\mathsf{op} = (\mathtt{removenode}, N)$, we replace the Leave operation with $\mathsf{Sim}(\mathtt{removenode}, N, \kappa)$ and set $b_\ell = 1$ for all the put labels that have been seen in the past. Similarly if $\mathsf{op} = (\mathtt{addnode}, N)$, we replace the Join operation with $\mathsf{Sim}(\mathtt{addnode}, N, \kappa)$ 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 $(\varepsilon, \delta, \theta)$-stable, with probability at least $1-\delta$, for all $\mathbf{C} \subseteq \mathbf{N}$, $(\omega, \mathbf{C})$ will be (ε, θ)-balanced. 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 5.1, 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 5.1. 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 5.1, 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 $\Pr\left[\mathbf{view_2}^i(I') = \mathbf{v}\right]$ is equal to

$$\sum_{(\alpha_1,\ldots,\alpha_q)\in\mathbf{A}^q} \Pr\left[\mathbf{view_2}^i(I') = \mathbf{v} \mid A_r = (\alpha_1, \ldots, \alpha_q)\right] \cdot \Pr\left[A_r = (\alpha_1, \ldots, \alpha_q)\right] \tag{12}$$

Similarly, let A_s be the random variable denoting the allocation of $t_1, \ldots, t_q$ to addresses in $\mathbf{view_3}$. Then, $\Pr\left[\mathbf{view_3}^i(I') = \mathbf{v}\right]$

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

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,

$$\Pr\left[\mathbf{view_2}^i(I') = \mathbf{v} \mid A_r = (\alpha_1, \ldots, \alpha_q)\right] = \Pr\left[\mathbf{view_3}^i(I') = \mathbf{v} \mid A_s = (\alpha_1, \ldots, \alpha_q)\right] \tag{13}$$

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 [2]

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

[2] 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.

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, $\alpha_j \in \mathsf{Vis}(\mathsf{fe}(t_j), I')$. 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 $a_j \leftarrow \Delta_1(\mathsf{Vis}(\mathsf{fe}(t_j), I'))$. Now,

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

In the second case, $\alpha_j \in \mathbf{A} \setminus \mathsf{Vis}(\mathsf{fe}(t_j), I')$. Then,

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

where

$$\begin{aligned}\Pr\left[\, \mathcal{E}_1 \,\right] &= \Pr\left[\, b_j = 1 \wedge b'_j = 0 \wedge a_j = \alpha_j \,\right], \text{ and} \\ \Pr\left[\, \mathcal{E}_2 \,\right] &= \Pr\left[\, b_j = 0 \wedge a_j = \alpha_j \,\right]\end{aligned}$$

such that $a_j \leftarrow \Delta_1(\mathbf{A} \setminus \mathsf{Vis}(\mathsf{fe}(t_j), I'))$. Let $B = \mathsf{Vis}(\mathsf{fe}(t_j), I')$, and let $G = \mathbf{A} \setminus \mathsf{Vis}(\mathsf{fe}(t_j), I')$. Then,

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

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

Adding the two probabilites, we get,

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

Hence,

$$\Pr\left[\, A_r = (\alpha_1, \ldots, \alpha_q) \,\right] = \Pr\left[\, A_s = (\alpha_1, \ldots, \alpha_q) \,\right] \tag{14}$$

Therefore, by substituting Equations 13 and 14 in Equation 12, we conclude that at the first churn operation,

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

Moreover, since the allocation distribution before the churn operation is the same and both the games use the same $\mathsf{DHT.Put}$ 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. □

Security of Chord-based BDX^+. In the following Corollary we formally state the security of BDX^+ scheme when its underlying DHT is instantiated with transient Chord. The proof follows directly from the fact that Chord has non-committing allocations and from Theorem 4.6, which shows that Chord is stable for:

$$\varepsilon = \frac{6(\theta + d)\log|\mathbf{N}|}{|\mathbf{N}|} + \frac{2\theta \log|\mathbf{C}|}{|\mathbf{C}|} \quad \text{and} \quad \theta \leq 4|\mathbf{N}|e^{-2} - d \quad \text{and} \quad \delta = 1 - o\left(\frac{1}{|\mathbf{N}|}\right)$$

Corollary 7.2. *If $|\mathbf{L}| = \Theta(2^k)$, $|I| = \theta \leq 4ne^{-2} - d$, and if EDDX^+ is instantiated with transient Chord, then it is $\mathcal{L}_\varepsilon$-secure with probability at least $1 - o(\frac{1}{|\mathbf{N}|}) - \mathsf{negl}(k)$ in the random oracle model, where*

$$\varepsilon = \frac{6(\theta + d)\log|\mathbf{N}|}{|\mathbf{N}|} + \frac{2\theta \log|\mathbf{C}|}{|\mathbf{C}|}.$$

Efficiency. The time, round and communication complexities of add and remove node operations of the BDX^+ are the same as the underlying DHT.

8 Conclusion

In this work, we initiated the study of distributed encrypted data structures and of DDXs in particular. We designed encrypted DDXs in both the perpetual and the transient settings. Our constructions used DHTs as a building block and we analyzed their security guarantees in terms of the load balancing properties of the underlying DHTs. We also analyzed their security when the underlying DHT is instantiated with Chord. We see our work as a first step towards designing provably-secure end-to-end encrypted distributed systems like end-to-end encrypted databases, off-chain networks, distributed storage systems, and distributed caches. Our work motivates several open problems and directions for future work.

Beyond Chord. The most immediate direction is to study the security of BDX when it is instantiated with other DHTs than Chord. Instantiations based on Kademlia [30] and Koorde [21] would be particularly interesting due to the former's popularity in practice and the latter's theoretical efficiency. Because Koorde is similar to Chord in structure (though its routing is different and based on De Bruijn graphs) the bounds we introduce in this work to study Chord's balance and stability might find use in analyzing Koorde. Kademlia, on the other hand, has a very different structure than Chord so it is likely that new custom techniques and bounds are needed to analyze its balance and stability.

New EDDX constructions. Another direction is to design new EDDX schemes with better leakage profiles. Here, a "better" profile could be the same profile $\mathcal{L}_\varepsilon$ achieved in this work but with a smaller ε than what we show. Alternatively, it could be a completely different leakage profile. This might be done, for example, by using more sophisticated techniques from cryptography, (e.g., leakage suppression, oblivious RAMs) and distributed systems (e.g., replication, consensus).

Encryption in other distributed data structures. A third immediate direction is to design and analyze end-to-end encryption in other distributed data structures such as multimaps, trees and graphs. Since data structures are the most basic building blocks of any system, getting encryption into them would enable us to build more sophisticated privacy-preserving systems on top of them. Moreover, the process of integrating encryption into the most basic distributed data structures would help us deepen our understanding of the impact the properties of distributed systems have on secruity.

Adding replication for reliability. Another important direction of immediate practical interest is to design a *replicated* encrypted DDX. In such a system, the encrypted label/value pairs would be replicated throughout the network for reliability. Replication, however, introduces a host of challenges including the problem of updating pairs in a consistent manner. Consistent distributed systems are already very hard to design (and prove correct) but the added challenge of achieving consistency on end-to-end encrypted data would seemingly make the problem even harder. The benefit, however, is that such a system would essentially yield a provably-secure end-to-encrypted reliable database which, in of itself, would be impactful.

Connections between consistency notions and security. When multiple operations execute concurrently on a distributed data structure, the output of an operation is not fixed and is determined by the *consistency* guarantee offered by how the data structure is designed. This means that different implementations of a data structure offer different consistency guarantees. One can look at different notions of consistencies as trade-offs between performance and correctness. While cryptography has provided us with a reasonable understanding of the relationship between leakage and efficiency, and distributed systems has provided us with a good understanding of the relationship between consistency and efficiency, it is not clear how consistency and leakage interact with each other. In particular, are stronger notions of consistency better for security or vice versa? It is a particularly intriguing question which would require us to understand how the two fields interact with each other.

Stronger adversarial models. Our security definitions are in the standalone model and against an adversary that makes static corruptions. Extending our work to handle arbitrary compositions (e.g., using universal composability [9]) and adaptive corruptions would be very interesting.

References

[1] Queryable encryption. https://www.mongodb.com/docs/manual/core/queryable-encryption/.

[2] Bittorrent, Accessed May, 2022. https://www.bittorrent.com/.

[3] A next-generation smart contract and decentralized application platform, Accessed May, 2022. https://github.com/ethereum/wiki/wiki/White-Paper.

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

[5] R. Bost. Sophos - forward secure searchable encryption. In *ACM Conference on Computer and Communications Security (CCS '16)*, 20016.

[6] R. Bost, B. Minaud, and O. Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *ACM Conference on Computer and Communications Security (CCS '17)*, 2017.

[7] 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.

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

[9] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[10] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO '13*. Springer, 2013.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos Garofalakis. Practical private range search revisited. In *Proceedings of the 2016 International Conference on Management of Data*, pages 185–198. ACM, 2016.

[18] 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.

[19] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. In *Computer Security–ESORICS 2015: 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II 20*, pages 123–145. Springer, 2015.

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

[21] 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.

[22] S. Kamara and T. Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Advances in Cryptology - EUROCRYPT '17*, 2017.

[23] S. Kamara and T. Moataz. SQL on Structurally-Encrypted Data. In *Asiacrypt*, 2018.

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

[25] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM Conference on Computer and Communications Security (CCS '12)*. ACM Press, 2012.

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

[27] 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.

[28] Protocol Labs. Filecoin: A decentralized storage network, Accessed May, 2022. `https://filecoin.io/filecoin.pdf`.

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

[30] 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.

[31] X. Meng, S. Kamara, K. Nissim, and G. Kollios. Grecs: Graph encryption for approximate shortest distance queries. In *ACM Conference on Computer and Communications Security (CCS 15)*, 2015.

[32] 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.

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

[34] 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.

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

[36] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 329–350. Springer, 2001.

[37] 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.

[38] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symposium on Research in Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

[39] 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.

[40] 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.

[41] 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.

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

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