Encrypted Key-Value Stores

Archita Agarwal[1] and Seny Kamara[1]

Brown University {archita_agarwal, seny_kamara}@brown.edu

Abstract. Distributed key-value stores (KVS) are distributed databases that enable fast access to data distributed across a network of nodes. Prominent examples include Amazon's Dynamo, Facebook's Cassandra, Google's BigTable and LinkedIn's Voldemort. The design of secure and private key-value stores is an important problem because these systems are being used to store an increasing amount of sensitive data. Encrypting data at rest and decrypting it before use, however, is not enough because each decryption exposes the data and increases its likelihood of being stolen. End-to-end encryption, where data is kept encrypted at all times, is the best way to ensure data confidentiality.

In this work, we study end-to-end encryption in distributed KVSs. We introduce the notion of an encrypted KVS and provide formal security definitions that capture the properties one would desire from such a system. We propose and analyze a concrete encrypted KVS construction which can be based on any unencrypted KVS. We first show that this construction leaks at most the operation equality (i.e., if and when two unknown queries are for the same search key) which is standard for similar schemes in the non-distributed setting. However, we also show that if the underlying KVS satisfies *read your writes* consistency, then the construction only leaks the operation equality of search keys that are handled by adversarially corrupted nodes—effectively showing that a certain level of consistency can improve the security of a system. In addition to providing the first formally analyzed end-to-end encrypted key-value store, our work identifies and leverages new and interesting connections between distributed systems and cryptography.

1 Introduction

A distributed key-value store (KVS) is a distributed storage system that stores label/value [1] pairs and supports get and put queries. KVSs provide one of the simplest data models but have become fundamental to modern systems due to their high performance, scalability and availability. For example, some of the largest social networks, e-commerce websites, cloud services and community forums depend on key-value stores for their storage needs. Prominent examples of distributed KVSs include Amazon's Dynamo [25], Facebook's Cassandra [43], Google's BigTable [22], LinkedIn's Voldemort [52], Redis [5], MemcacheDB [4] and Riak [53].

[1] In this work we use the term *label* and reserve the term *key* to denote cryptographic keys.

Distributed KVSs are closely related to distributed hash tables (DHT) and, in fact, most are built on top of a DHT. However, since DHTs do not necessarily guarantee fault-tolerance, KVSs use various techniques to achieve availability in the face of node failures. The simplest approach is to replicate each label/value pair on multiple nodes and to use a replica control protocol to guarantee some form of consistency.

End-to-end encryption in KVSs. As an increasing amount of data is being stored and managed by KVSs, their security has become an important problem. Encryption is often proposed as a solution, but encrypting data in transit and at rest and decrypting it before use is not enough since each decryption exposes the data and increases its likelihood of being stolen. A better way to protect data is to use end-to-end encryption where a data owner encrypts its data with its own secret key (that is never shared). End-to-end encryption guarantees that data is encrypted at all times—even in use—which ensures data confidentiality.

Our contributions. In this work, we formally study the use of end-to-end encryption in KVSs. In particular, we extend the recently proposed framework of Agarwal and Kamara [7] from DHTs to KVSs. We formalize the goals of encryption in KVSs by introducing the notion of an *encrypted key-value store* (EKVS) and propose formal syntax and security definitions for these objects. The simplest way to design an EKVS is to store label/value pairs (ℓ, v) as $(F_{K_1}(\ell), \mathsf{Enc}_{K_2}(v))$ in a standard/plaintext KVS, where F is a pseudo-random function and Enc is a symmetric encryption scheme. The underlying KVS will then replicate the encrypted pair, store the replicas on different storage nodes, handle routing, node failures and consistency. Throughout, we will refer to this approach as the *standard scheme* and we will use our framework to formally study its security properties. We make the following contributions:

- *formalizing KVSs*: we provide an abstraction of KVSs that enables us to isolate and analyze several important properties of standard/plaintext KVSs that impact the security of the standard EKVS. More precisely, we find that the way a KVS distributes its data and the extent to which it load balances have a direct effect on what information an adversary can infer about a client's queries.
- *distributed leakage analysis*: an EKVS can be viewed as a distributed version of an encrypted dictionary which is a fundamental building block in the design of sub-linear encrypted search algorithms (ESA). All sub-linear ESAs leak some information—whether they are built from property-preserving encryption, structured encryption or oblivious RAMs—so our goal is to identify and prove the leakage profile of the standard scheme. Leakage analysis in the distributed setting is particularly challenging because the underlying distributed system (in our case the underlying KVS) can create very subtle correlations between encrypted data items and queries. As we will see, replication makes this even more challenging. We consider two cases: the single-user case where the EKVS stores the datasets of multiple clients but

each dataset can only be read and updated by its owner; and the multi-user case where each dataset can be read and updated by multiple users.

– *leakage in the multi-user case*: We show that in the multi-user setting, the standard scheme leaks the operation equality (i.e., if and when get and put operations are for the same label) over all operations; even operations that are not handled by corrupted nodes. [2] This may seem surprising since it is not clear a-priori why an adversary would learn anything about data that it never "sees".
– *leakage in the single-user case*: In the single-user scenario, we show that, if the standard scheme's underlying KVS achieves read your write (RYW) consistency, then it only leaks the operation equality over operations that are handled by corrupted nodes. This is particularly interesting as it suggests that stronger consistency guarantees improve the security of end-to-end encrypted KVSs.
– *comparison with DHTs*: As mentioned earlier, the main difference between a DHT and a KVS is that the latter replicate data on multiple nodes. To ensure a consistent view of this data, KVSs need to implement some consistency model. Achieving strong consistency, however, is very costly so almost all practical systems achieve weaker notions which cannot guarantee that a unique value will always be associated to a given label. In particular, the value that will be returned will depend on factors such as network delay, synchronization policy and the ordering of concurrent operations. Therefore, an adversary that controls one or more of these factors can affect the outputs of a KVS. It therefore becomes crucial to understand and analyze this correlation when considering the security of an encrypted KVS. In contrast, this is not needed in the case of encrypted DHTs since they do not maintain replicas and hence consistency is not an issue.
– *concrete instantiations*: We use our framework to study two concrete instantiations of the standard scheme. The first uses a KVS based on consistent-hashing with zero-hop routing whereas the second uses a KVS based on consistent hashing with multi-hop routing.

2 Related Work

Key-value stores. NoSQL databases were developed as an alternative to relational databases to satisfy the performance and scalability requirements of large Internet complanies. KVSs are the simplest kind of NoSQL databases. Even though such databases had already existed, they gained popularity when Amazon developed Dynamo [25], a KVS for its internal use. Since then many KVSs have been developed both in industry and academia. Most prominent ones are Facebook's Cassandra [43], Google's BigTable [22], LinkedIn's Voldemort [52], Redis [5], MemcacheDB [4] and Riak [53]. All of them are eventually consistent but some of them can be tuned to provide strong consistency [53, 5, 43]. There

[2] Note that the operation equality is a common leakage pattern in practical ESAs.

have also been efforts to develop KVSs with stronger consistency such as causal consistency [44, 45, 54, 11], and strong consistency [2, 1, 3, 6].

Encrypted search. An encrypted search algorithm (ESA) is a search algorithm that operates on encrypted data. ESAs can be built from various cryptographic primitives including oblivious RAM (ORAM) [33], fully-homomorphic encryption (FHE) [31], property-preserving encryption (PPE) [8, 12, 14, **?**, **?**] and structured encryption (STE) [23] which is a generalization of searchable symmetric encryption [49]. Each of these approaches achieves different tradeoffs between efficiency, expressiveness and security/leakage. For large datasets, structured encryption seems to provide the best tradeoffs between these three dimensions: achieving sub-linear (and even optimal) search times and rich queries while leaking considerably less than PPE-based solutions and either the same as [39] or slightly more than ORAM-based solutions. Various aspects of STE have been extensively studied in the cryptographic literature including dynamism [32, 42, 41, 50, 15, 19, 16, 28], locality [21, 9, 27, 10, 26], expressiveness [23, 20, 48, 30, 29, 47, 36, 37, 40, **?**] and leakage [34, 18, 39, 13, 38]. Encrypted key-value stores can be viewed as a form of distributed STE scheme. Such schemes were first considered by Agarwal and Kamara in [7] where they studied encrypted distributed hash tables.

3 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 $x \leftarrow \mathcal{A}$. If S is a set then $|S|$ refers to its cardinality. If s is a string then $|s|$ 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 $\{(\ell_i, v_i)\}_{i\le n}$ 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.

Leakage profiles. Many cryptographic primitives and protocols leak information. Examples include encryption schemes, which reveal the length of the plaintext; secure multi-party computation protocols, which (necessarily) reveal about the parties' inputs whatever can be inferred from the output(s); order-preserving encryption schemes, which reveal implicit and explicit bits of the plaintext; structured encryption schemes which reveal correlations between queries; and oblivious algorithms which reveal their runtime and the volume of data they read. Leakage-parameterized security definitions [24, 23] extend the standard provable security paradigm used in cryptography by providing adversaries (and simulators) access to leakage over plaintext data. This leakage is formally and precisely captured by a leakage profile which can then be analyzed through cryptanalysis and further theoretical study. Leakage profiles can themselves be functions of one or several leakage patterns. Here, the only pattern we will consider is the *operation equality* which reveals if and when two (unknown) operations are for the same label.

Consistency guarantees. The consistency guarantee of a distributed system specifies the set of acceptable responses that a read operation can output. There are multiple consistency guarantees studied in the literature, including *linearizability, sequential consistency, causal consistency and eventual consistency.* Though strong consistency notions like linearizability are desirable, the Consistency, Availability, and Partition tolerance (CAP) Theorem states that strong consistency and availability cannot be achieved simultaneously in the presence of network partitions. Therefore, many practical systems settle for weaker consistency guarantees like sequential consistency, causal consistency and eventual consistency. We note that all these weaker consistency guarantees—with the exception of eventual consistency—all satisfy what is known as "Read Your Writes" (RYW) consistency which states that all the writes performed by a single client are visible to its subsequent reads. Many practical systems [5, 44, 45, 2, 1] guarantee RYW consistency.

4 Key-Value Stores

Here we extend the formal treatment of encrypted DHTs given by Agarwal and Kamara [7] to key-value stores. A key-value store is a distributed storage system that provides a key-value interface and that guarantees *resiliency* against node failures. It does so by replicating label/value pairs on multiple nodes. Similar to DHTs, there are two kinds of KVSs: perpetual and transient. Perpetual KVSs are composed of a fixed set of nodes that are all known at setup time. Transient KVSs, on the other hand, are designed for settings where nodes are not known a-priori and can join and leave at any time. Perpetual KVSs are suitable for "permissioned" settings like the backend infrastructure of large companies whereas transient KVSs are better suited to "permissionless" settings like peer-to-peer networks and permissionless blockchains. In this work, we study the security of pertpetual KVSs.

Perpetual KVSs. We formalize KVSs as a collection of six algorithms $\mathsf{KVS} = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{FrontEnd}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$. The first three algorithms, $\mathsf{Overlay}$, Alloc and $\mathsf{FrontEnd}$ are executed only once by the entity responsible for setting up the system. $\mathsf{Overlay}$ takes as input an integer $n \geq 1$, and outputs a parameter ω from a space Ω. Alloc takes as input parameters ω, n and an integer $\rho \geq 1$, and outputs a parameter ψ from a space Ψ. $\mathsf{FrontEnd}$ takes as input parameters ω and n and outputs a parameter ϕ from space Φ. Intuitively, the parameter ϕ will be used to determine a *front end* node for each label. These front end nodes will serve as the clients' entry points in the network whenever they need perform an operation on a label. We refer to ω, ψ, ϕ as the *KVS parameters* and represent them by $\Gamma = (\omega, \psi, \phi)$. Each KVS has an address space $\mathbf{A}$ and the KVS parameters in Γ define different components of the KVS over this address space. For example, ω maps node names to addresses in $\mathbf{A}$, ψ maps labels to addresses in $\mathbf{A}$, ϕ determines the address of a front-end node (or starting node). The fourth algorithm, Daemon, takes Γ and n as input and is executed by every node in the network. Daemon is halted only when a node wishes to leave the network and it is responsible for setting up its calling node's state for routing messages and for storing and retrieving label/value pairs from the node's local storage. The fifth algorithm, Put, is executed by a client to store a label/value pair on the network. Put takes as input Γ and a label/value pair ℓ and v, and outputs nothing. The sixth algorithm, Get, is executed by a client to retrieve the value associated to a given label from the network. Get takes as input Γ and a label ℓ and outputs a value v. Since all KVS algorithms take Γ as input we sometimes omit it for visual clarity.

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

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

where addr_ω maps node names from a name space $\mathbf{N}$ to addresses from an address space $\mathbf{A}$, $\mathsf{replicas}_{\omega,\psi}$ maps labels from a label space $\mathbf{L}$ to the set of addresses of ρ nodes that store it, route_ω maps two addresses to the addresses of the nodes on the route between them, and fe_ϕ maps labels to node addresses who forward client requests to the rest of the network. [3] 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. Given an address a and set of addresses S, we also sometimes write $\mathsf{route}_\omega(a, S)$ to represent the set of routes from a to all the addresses in S. Note that this is an abstract representation of a KVS that will be particularly useful to us to define random variables we need for our probabilistic analysis but, in practice, the overlay network, including its addressing and routing functions, are implemented by the Daemon algorithm.

We sometimes refer to a pair $(\omega, \mathbf{C})$ as an overlay and to a pair $(\psi, \mathbf{K})$ as an allocation. Abstractly speaking, we can think of an overlay as an assignment from active nodes to addresses and of an allocation as an assignment of active

[3] For KVSs that allow their clients to connect directly to the replicas and do not use front end nodes, the abstraction can drop the fe mapping and be adjusted in the natural way.

labels to addresses. In this sense, overlays and allocations are determined by a pair $(\omega, \mathbf{C})$ and $(\psi, \mathbf{K})$, respectively. [4]

Visible addresses. As in [7], a very useful notion for our purposes will be that of *visible addresses.* For a fixed overlay $(\omega, \mathbf{C})$ and a fixed replication parameter ρ, an address $a \in \mathbf{A}$ is $\mathbf{s}$-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) $\mathsf{addr}_\omega(N) \in \mathsf{replicas}_{\omega,\psi}(\ell)$; or (2) $\mathsf{addr}_\omega(N) \in \mathsf{route}_\omega(s, \mathsf{replicas}_{\omega,\psi}(\ell))$. The intuition behind this is that if a label ℓ is mapped to an address in $\mathsf{Vis}(s, N)$ then N either stores the label ℓ or routes it when the operation for ℓ starts at address s. We point out that the visibility of a node changes as we change the starting address s. For example, the node maybe present on the path to one of the addresses if s was the starting address but not on the path if some other address s' was the starting address. Throughout we assume the set of visible addresses to to be efficiently computable.

Since the set of $\mathbf{s}$-visible addresses depends on parameters ω and ρ, and the set $\mathbf{C}$ of nodes that are currently active, we subscript $\mathsf{Vis}_{\omega,\mathbf{C},\rho}(s, N)$ with all these paramters. Finally, as in [7], we also extend the notion to the set of s-visible addresses $\mathsf{Vis}_{\omega,\mathbf{C},\rho}(s, S)$ for a set of nodes $S \subseteq \mathbf{C}$ which is defined simply as $\mathsf{Vis}_{\omega,\mathbf{C},\rho}(s, S) = \cup_{N \in S} \mathsf{Vis}_{\omega,\mathbf{C},\rho}(s, N)$. Again, for visual clarity, we will drop the subscripts wherever they are clear from the context.

Front-end distribution. As in [7], another important notion in our analysis is that of a label's *front-end distribution* which is the probability distribution that governs the address of an operation's "entry point" into the KVS network. It is captured by the random variable $\mathsf{fe}_\phi(\ell)$, where ϕ is sampled by the algorithm $\mathsf{FrontEnd}$. In this work we assume front-end distributions to be *label-independent* in the sense that every label's front-end node distribution is the same. We therefore simply refer to this distribution as the KVS's front-end distribution.

Allocation distribution. The next notion important to our analysis is what we refer to as a label's *allocation distribution* which is the probability distribution that governs the address at which a label is allocated. More precisely, this is captured by the random variable $\psi(\ell)$, where ψ is sampled by the algorithm Alloc. In this work, we assume allocation distributions are *label-independent* in the sense that every label's allocation distribution is the same. We refer to this distribution as the KVS's allocation distribution. [5]

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

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

[4] Note that for simplicity, we assume that ψ maps labels to a single address. This however can be extended in a straightforward way where ψ maps a label to multiple addresses. This would be required to model KVSs where replicas of a label are independent of each other.

[5] This is true for every KVS we are aware of [25, 43, 52, 53].

where f_ψ is the probability mass function of the KVS's allocation distribution.

Non-committing allocations. As we will see in Section 6, our EKVS construction can be based on any KVS but the security of the resulting scheme will depend on certain properties of the underlying KVS. We describe these properties here. The first property that we require of a KVS 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 $(\psi, \mathbf{K})$, there should exist some efficient mechanism to arbitrarily change/program ψ. In other words, there should exist a polynomial-time algorithm $\mathsf{Program}$ such that, for all $(\omega, \mathbf{C})$ and $(\psi, \mathbf{K})$, given a label $\ell \in \mathbf{L}$ and an address $a \in \mathbf{A}$, $\mathsf{Program}(\ell, a)$ modifies the KVS so that $\psi(\ell) = a$. For the special case of consistent hashing based KVSs, which we study in Section 7, this can be achieved by modeling one of its hash functions as a random oracle.

Balanced overlays. The second property is related to how well the KVS 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, that any set of θ nodes sees ℓ is not too large.

Definition 1 (Balanced overlays). *Let $\omega \in \Omega$ be an overlay parameter, $\mathbf{C} \subseteq \mathbf{N}$ be a set of active nodes, and $\rho \geq 1$ be a replication parameter. 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 $|S| = \theta$,*

$$\Pr\left[\, \mathsf{replicas}_{\omega,\psi}(\ell) \cap \mathsf{Vis}_{\omega,\mathbf{C},\rho}(\mathsf{fe}_\phi(\ell), S) \neq \emptyset \,\right] \leq \varepsilon,$$

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

Definition 2 (Balanced KVS). *We say that a key-value store* $\mathsf{KVS} = (\mathsf{Overlay}, \mathsf{Alloc}, \mathsf{FrontEnd}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$ *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* $\mathsf{Overlay}$ *and where ε and δ can depend on $\mathbf{C}$ and θ.*

5 Encrypted Key-Value Stores

In this Section, we formally define encrypted key-value stores. An EKVS is an end-to-end encrypted distributed system that instantiates a replicated dictionary data structure.

5.1 Syntax and Security Definitions

Syntax. We formalize EKVSs as a collection of seven algorithms $\mathsf{EKVS} = (\mathsf{Gen}, \mathsf{Overlay}, \mathsf{Alloc}, \mathsf{FrontEnd}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$. The first algorithm Gen is executed by a client and takes as input a security parameter 1^k and outputs a secret key

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

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

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

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

K. All the other algorithms have the same syntax as before (See Section 4), with the difference that Get and Put also take the secret key K as input.

Security. The definition is roughly the same as the one in [7] and is based on 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 shown in Figure 1. The protocol is secure if no environment can distinguish between the outputs of these two experiments.

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. Due to space constraints, we detail both the experiments more formally in the full version of the paper.

Definition 3 ($\mathcal{L}$-security). *We say that an encrypted key-value store* $\mathsf{EKVS} = (\mathsf{Gen}, \mathsf{Overlay}, \mathsf{Alloc}, \mathsf{FrontEnd}, \mathsf{Daemon}, \mathsf{Put}, \mathsf{Get})$ *is $\mathcal{L}$-secure, if for all* PPT *adversaries $\mathcal{A}$ and all* PPT *environments $\mathcal{Z}$, there exists a* PPT *simulator* Sim *such that for all $z \in \{0,1\}^*$,*

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

Correctness. In the real/ideal-world paradigm, the security of a protocol is tied to its correctness. It is therefore important that our ideal functionality capture the correctness of the KVS as well. What this means is that the functionality should produce outputs that follow the same distribution as the outputs from a KVS. Unfortunately, in a setting with multiple clients sharing the data, even with the strongest consistency guarantees (e.g., linearizability), there are multiple possible responses for a read, and the one which the KVS actually outputs depends on behaviour of the network. Since the network behaviour is non-deterministic, the distribution over the possible outputs is also non-deterministic and hence the functionality cannot model the distribution over outputs correctly without modelling the network inside it.

However, if we restrict to a single client setting, RYW property ensures that a Get always outputs the latest value written to the KVS. Therefore the functionality $\mathcal{F}_{\mathsf{KVS}}$ models the correct distribution over the outputs: on a $\mathsf{Get}(\ell)$, it outputs the last value written to $\mathsf{DX}[\ell]$, and on a $\mathsf{Put}(\ell, v)$, it updates the $\mathsf{DX}[\ell]$ to v.

6 The Standard EKVS Scheme in the Single-User Setting

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

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

The Gen algorithm takes as input a security parameter 1^k and uses it to generate a key K_1 for the pseudo-random function F and a key K_2 for the symmetric encryption scheme SKE. It then outputs a key $K = (K_1, K_2)$. The $\mathsf{Overlay}$, Alloc, $\mathsf{FrontEnd}$ and Daemon algorithms respectively execute $\mathsf{KVS.Overlay}$, $\mathsf{KVS.Alloc}$, $\mathsf{KVS.FrontEnd}$ and $\mathsf{KVS.Daemon}$ to generate and output the paramters ω, ψ and ϕ. The Put algorithm takes as input the secret key K and a label/value pair (ℓ, v). It first computes $t := F_{K_1}(\ell)$ and $e \leftarrow \mathsf{Enc}(K_2, v)$ and then executes $\mathsf{KVS.Put}(t, e)$. The Get algorithm takes as input the secret key K and a label ℓ. It computes $t := F_{K_1}(\ell)$ and executes $e \leftarrow \mathsf{KVS.Get}(t)$. It then outputs $\mathsf{SKE.Dec}(K, e)$.

Security. We now describe the leakage of EKVS. 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 $\ell$ 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.

Discussion. We now explain why the leakage function is probabilistic and why it depends on the balance of the underlying KVS. Intuitively, one expects that

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

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

Fig. 2. The Standard EKVS Scheme

the adversary's view is only affected by get and put operations on labels that are either: (1) allocated to a corrupted node; or (2) allocated to an uncorrupted node whose path 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 by the choice of overlay, allocation and front-end function—all of which are chosen at random. The way this is handled in the leakage function is by sampling a bit b with some probability and revealing leakage on the current operation if $b = 1$. This determines the subset of operations whose leakage will be visible to the adversary.

Now, for the simulation to go through, the operations simulated by the simulator need to be visible to the adversary with the same probability as in the real execution. But these probabilities depend on ω, ψ and ϕ 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, the allocation and front-end function of the KVS.

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 KVS 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 whose proof is in the full version of the paper.

Theorem 1. *If $|I| \leq \theta$ and if* KVS *is RYW consistent, $(\varepsilon, \delta, \theta)$-balanced, has non-committing allocations and has label-independent allocation and front-end distributions, then* EKVS *is $\mathcal{L}_\varepsilon$-secure with probability at least* $1 - \delta - \mathsf{negl}(k)$.

Efficiency. The standard scheme does not add any overhead to time, round, communication and storage complexities of the underlying KVS.

7 A Concrete Instantiation Based on Consistent Hashing

In this section, we analyze the security of the standard EKVS when its underlying KVS is instantiated with a consistent hashing based KVS (CH-KVS). We first give a brief overview of consistent hashing and then show that: (1) it has non-committing allocations in the random oracle model; and (2) it is balanced under two commonly used routing protocols.

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

The parameter space Ψ is the set of all hash functions $\mathcal{H}_2$ from $\mathbf{L}$ to $\mathbf{A} = \{0, \ldots, 2^m - 1\}$. Alloc samples a hash function H_2 uniformly at random from $\mathcal{H}_2$ and outputs $\psi = H_2$. The map $\mathsf{replicas}_{\omega,\psi}$ maps every label ℓ in $\mathbf{L}$ to the addresses of ρ active nodes that follow $H_2(\ell)$ in clockwise direction. More formally, $\mathsf{replicas}_{\omega,\psi}$ is the mapping $(\mathsf{succ}_{\chi_\mathbf{C}} \circ H_2, \ldots, \mathsf{succ}^\rho_{\chi_\mathbf{C}} \circ H_2)$, where $\mathsf{succ}_{\chi_\mathbf{C}}$ is the *successor* function that assigns each address in $\mathbf{A}$ to its least upper bound in $\chi_\mathbf{C}$. Here, $\{0, \ldots, 2^m - 1\}$ is viewed as a "ring" in the sense that the successor of 2^{m-1} is 0.

CH-KVSs allow their clients to choose any node as the front-end node to issue its operations. Moreover, they do not restrict them to connect to the same node $\mathsf{fe}_\phi(\ell)$, everytime the client wants to query the same ℓ. This means that for

CH-KVSs, fe_ϕ is not necessarily a function but can be a one-to-many relation. Unfortunately we cannot prove CH-KVSs to be balanced for arbitrary fe_ϕs. We therefore modify CH-KVSs and model their space $\varPhi$ as the set of all hash functions $\mathcal{H}_3$ from $\mathbf{L}$ to addresses of active nodes. $\mathsf{FrontEnd}$ samples a hash function H_3 uniformly at random from $\mathcal{H}_3$ and outputs $\phi = H_3$. The map fe_ϕ is the hash function H_3 itself so it assigns a front-end node with address $H_3(\ell)$ to each label ℓ.

Routing protocols. There are two common routing protocols with CH-KVSs; each with trade-offs in storage and efficiency.

- *Multi-hop routing.* Based on H_1, the Daemon algorithm constructs a routing table by storing the addresses of the node's 2^ith successors where $0 \leq i \leq \log n$ (we refer the reader to [51] for more details). Note that a routing table contains at most $\log n$ other nodes. The routing protocol is fairly simple: given a message destined to a node N_d, a node N checks if $N = N_d$. If not, the node forwards the message to the node N' in its routing table with an address closest to N_d. Note that the route_ω map is deterministic given a fixed set of active nodes and it guarantees that any two nodes have a path of length at most $\log n$.
- *Zero-hop routing.* Based on H_1, the Daemon algorithm constructs a routing table by storing the addresses of all the other nodes in the routing table. Routing is then straightforward: given a message for N_d, simply forward it to the address of N_d. In short, for any two addresses s and d, $\mathsf{route}_\omega(s, d) = \{s, d\}$.

Storing and retrieving. When a client wants to execute a Get/Put operation on a label ℓ, it forwards the operation to the front-end node of ℓ. The front-end node executes the operation on the client's behalf as follows. It computes $\mathsf{replicas}(\ell)$ and forwards the operation to one of them. This replica is called the coordinator node. The coordinator then sends the operation to all (or a subset) the other replicas which then either update their state (on Put) or return a response back to the coordinator (on Get). In case more than one value is returned to the coordinator, it decides which value(s) is to be returned to the front-end. The choice of the coordinator node for a label ℓ varies from KVS to KVS. It can be a fixed node or a different node between requests for label ℓ. Either way, it is always a node chosen from the set of replicas. This guarantees that the visibility of a label (and hence the leakage) does not change between requests. KVSs also employ different synchronization mechanisms, like Merkle trees and read repairs to synchronize divergent replicas.

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

Allocation distribution. We now describe the allocation distribution of CH-KVSs. Since CH-KVSs assign labels to addresses using a random oracle H_2, it follows that for all overlays $(H_1, \mathbf{C})$, all labels $\ell \in \mathbf{L}$ and addresses $a \in \mathbf{A}$,

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

which implies that CH-KVSs have label-independent allocations. From this it also follows that $\Delta(S)$ has a probability mass function

$$f_{\Delta(S)}(a) = \frac{f_\psi(a)}{\sum_{a \in S} f_\psi(a)} = \frac{1}{|\mathbf{A}|}\left(\frac{|S|}{|\mathbf{A}|}\right)^{-1} = \frac{1}{|S|}.$$

Before describing the visibility of nodes in CH-KVSs and analyzing their balance under zero-hop and multi-hop routing protocols, we define notation that will be useful in our analysis.

Notation. The *arc* of a node N is the set of addresses in $\mathbf{A}$ between N's predecessor and itself. Note that the arc of a node depends on a configuration χ. More formally, we write $\mathsf{arc}_\chi(N) = (\mathsf{pred}_\chi(H_1(N)), \ldots, H_1(N)]$, where $\mathsf{pred}_\chi(N)$ is the *predecessor* function which assigns each address in $\mathbf{A}$ to its largest lower bound in χ. We extend the notion of arc of a node to ρ-arcs of a node. A ρ-arc of a node N is the set of addresses between N's ρth predecessor and itself. More formally, we write $\mathsf{arc}^\rho_\chi(N) = (\mathsf{pred}^\rho_\chi(H_1(N)), \ldots, H_1(N)]$, where $\mathsf{pred}^\rho_\chi(H_1(N))$ represents the predecessor function applied ρ times on $H_1(N)$. Intuitively, if H_2 hashes a label ℓ anywhere in ρ-arc of N, then N becomes one of the ρ replicas of ℓ. We denote by $\mathsf{maxareas}(\chi, x)$, the sum of the lengths (sizes) of x largest arcs in configuration χ. The *maximum area* of a configuration χ is equal to $\mathsf{maxareas}(\chi, \rho\theta)$. As we will later see, the maximum area is central to analyzing the balance of CH-KVSs.

7.1 Zero-hop CH-KVSs

In this section, we analyse the visibility and balance of zero-hop CH-KVSs.

Visible addresses. Given a fixed overlay $(H_1, \mathbf{C})$, an address $s \in \mathbf{A}$ and a node $N \in \mathbf{C}$, if the starting address is $s = H_1(N)$, then $\mathsf{Vis}_{\chi_\mathbf{C}}(s, N) = \mathbf{A}$. This is because $H_1(N)$ lies on $\mathsf{route}_{\chi_\mathbf{C}}(s, a)$ for all $a \in \mathbf{A}$. Now for an address $s \in \mathbf{A}$ such that $s \neq H_1(N)$, we have

$$\begin{aligned}
\mathsf{Vis}_{\chi_\mathbf{C}}(s, N) &= \left\{\mathsf{arc}^\rho_{\chi_\mathbf{C}}(N') : H_1(N) \in \mathsf{route}_{\chi_\mathbf{C}}(s, H_1(N'))\right\} \bigcup \mathsf{arc}^\rho_{\chi_\mathbf{C}}(N) \\
&= \left\{\mathsf{arc}^\rho_{\chi_\mathbf{C}}(N') : H_1(N) \in \{s, H_1(N')\}\right\} \bigcup \mathsf{arc}^\rho_{\chi_\mathbf{C}}(N) \\
&= \left\{\mathsf{arc}^\rho_{\chi_\mathbf{C}}(N') : H_1(N) = H_1(N')\right\} \bigcup \mathsf{arc}^\rho_{\chi_\mathbf{C}}(N) \\
&= \mathsf{arc}^\rho_{\chi_\mathbf{C}}(N)
\end{aligned}$$

where the second equality follows from the fact that $\mathsf{route}_{\chi_\mathbf{C}}(s, H_1(N')) = \{s, H_1(N')\}$, the third follows from the assumption that $H_1(N) \neq s$, and the fourth from the fact that $\mathsf{arc}^{\rho}_{\chi_{\mathbf{C}}}(N) = \mathsf{arc}^{\rho}_{\chi_{\mathbf{C}}}(N')$ if $H_1(N) = H_1(N')$. Finally, for any set $S \subseteq \mathbf{C}$, $\mathsf{Vis}_{\omega,\mathbf{C}}(s, S) = \cup_{N \in S} \mathsf{Vis}_{\omega,\mathbf{C}}(s, N)$.

Balance of zero-hop CH-KVSs. Before analyzing the balance of CH-KVSs, we first recall a Lemma from Agarwal and Kamara [7] that upper bounds the sum of the lengths of the x largest arcs in a configuration χ in Chord. The sum is denoted by $\mathsf{maxareas}(\chi, x)$. Since Chord is also based on consistent hashing, we use the corollary to bound the maximum area of CH-KVSs by substituting $x = \rho\theta$.

Lemma 1 ([7]). *Let $\mathbf{C} \subseteq \mathbf{N}$ be a set of active nodes. Then, for $x \leq |\mathbf{C}|/e$,*

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

We are now ready to analyze the balance of zero-hop CH-KVSs.

Theorem 2. *Let $\mathbf{C} \subseteq \mathbf{N}$ be a set of active nodes. If $\mathsf{maxareas}(\chi_{\mathbf{C}}, \rho\theta) \leq \lambda$, then $\chi_{\mathbf{C}}$ is (ε, θ)-balanced with*

$$\varepsilon = \frac{\theta}{|\mathbf{C}|} + \frac{\lambda}{|\mathbf{A}|}$$

The proof of Theorem 2 is in the full version of the paper.

Corollary 1. *Let $\mathbf{C}$ be a set of active nodes. For all $\rho\theta \leq |\mathbf{C}|/e$, a zero-hop CH-KVS is $(\varepsilon, \delta, \theta)$-balanced for*

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

The proof of Corollary 1 is in the full version of the paper.

Remark. It follows from Corollary 1 that

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

and $\delta = O(1/|\mathbf{C}|^2)$. Note that assigning labels uniformly at random to ρ nodes would achieve $\varepsilon = \rho\theta/|\mathbf{C}|$ so zero-hop CH-KVSs balance data fairly well.

The Security of a Zero-Hop CH-KVS based EKVS. In the following Corollary, we formally state the security of the standard scheme when its underlying KVS is instantiated with a zero-hop CH-KVS.

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

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

The proof of Corollary 2 is in the full version of the paper. From the discussion of Corollary 1, we know,

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

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

7.2 Multi-hop CH-KVSs

In this section, we analyse the visibility and balance of multi-hop CH-KVSs. Since most of the details are similar to what was in the last section, we keep the description high level.

Visible addresses. Given a fixed overlay $(H_1, \mathbf{C})$, an address $s \in \mathbf{A}$ and a node $N \in \mathbf{C}$, if the starting address is $s = H_1(N)$, then $\mathsf{Vis}_{\chi_\mathbf{C}}(s, N) = \mathbf{A}$. For an address $s \in \mathbf{A}$ such that $s \neq H_1(N)$, we have

$$\mathsf{Vis}_{\chi_\mathbf{C}}(s, N) = \left\{\mathsf{arc}^{\rho}_{\chi_\mathbf{C}}(N') : H_1(N) \in \mathsf{route}_{\chi_\mathbf{C}}(s, H_1(N'))\right\} \bigcup \mathsf{arc}^{\rho}_{\chi_\mathbf{C}}(N)$$

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

Balance of multi-hop CH-KVSs. We now analyze the balance of multi-hop CH-KVSs.

Theorem 3. *Let $\mathbf{C} \subseteq \mathbf{N}$ be a set of active nodes. If $\mathsf{maxareas}(\chi_\mathbf{C}, \rho\theta) \leq \lambda$, then $\chi_\mathbf{C}$ is (ε, θ)-balanced with*

$$\varepsilon = \frac{\rho\theta \log|\mathbf{C}|}{|\mathbf{C}|} + \frac{\lambda}{|\mathbf{A}|}$$

The proof of Theorem 3 is in the full version of the paper.

Corollary 3. *Let $\mathbf{C}$ be a set of active nodes. For all $\rho\theta \leq |\mathbf{C}|/(e\log|\mathbf{C}|)$, a multi-hop CH-KVS is $(\varepsilon, \delta, \theta)$-balanced for*

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

The Corollary follows directly from Corollary 1 and Theorem 3.
Notice that multi-hop CH-KVSs are not only less balanced than zero-hop CH-KVSs but also tolerate a lesser number of corruptions. This is the case because in a multi-hop CH-KVS there is a higher chance that an adversary sees a label since the routes are larger.

Remark. It follows from Corollary 3 that

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

and $\delta = O(1/|\mathbf{C}|^2)$. As discussed earlier, the optimal balance is $\varepsilon = \rho\theta/|\mathbf{C}|$, which is achieved when labels are assigned uniformly at random to ρ nodes. Note that balance of multi-hop CH-KVSs is only $\log|\mathbf{C}|$ factor away from optimal balance which is very good given that the optimal balance is achieved with no routing at all.

The Security of a Multi-Hop CH-KVS based EKVS. In the following Corollary, we formally state the security of the standard scheme when its underlying KVS is instantiated with a multi-hop CH-KVS.

Corollary 4. *If* $|\mathbf{L}| = \Theta(2^k)$, $|I| \leq |\mathbf{C}|/(\rho e \log|\mathbf{C}|)$, *and if* EKVS *is instantiated with a RYW multi-hop CH-KVS, then it is* $\mathcal{L}_\varepsilon$*-secure with probability at least* $1 - 1/|\mathbf{C}|^2 - (e^{-\sqrt{|\mathbf{C}|}} \cdot \log|\mathbf{C}|) - \mathsf{negl}(k)$ *in the random oracle model, where*

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

From the discussion of Theorem 3, we know that,

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

and $\delta = O(1/|\mathbf{C}|^2)$. Setting $|I| = |\mathbf{C}|/(\rho\alpha\log|\mathbf{C}|)$, for some $\alpha \geq e$, we have $\varepsilon = O(1/\alpha)$, which intuitively means that the adversary can expect to learn the operation equality of an $O(1/\alpha)$ fraction of client operations.

8 The Standard EKVS Scheme in the Multi-User Setting

We now analyze the security of the standard scheme in a more general setting, i.e., where we no longer require the underlying KVS to satisfy RYW and where we no longer assume that a single client operates on the data. We call this setting the *multi-user* setting where multiple clients operate on the same data concurrently. We start by extending our security definition to the multi-user setting and then analyze the security of the standard scheme (from Figure 2) in this new setting.

The ideal multi-user KVS functionality. The ideal multi-user KVS functionality $F^{\mathcal{L}}_{\mathsf{mKVS}}$ is described in Figure 3. The functionality stores all the values that were ever written to a label. It also associates a time τ with every value indicating when the value was written. On a Get operation, it sends leakage to the simulator which returns a time τ'. The functionality then returns the value

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

$F^{\mathcal{L}}_{\mathsf{mKVS}}$ stores a dictionary DX initialized to empty, a time counter τ initialized to 0 and proceeds as follows, running with c clients $\mathcal{C}_1, \ldots, \mathcal{C}_c$, n nodes $N_1, \ldots, N_n$ and a simulator Sim:

- $\mathsf{Put}(\ell, v)$: Upon receiving a label/value pair (ℓ, v) from some client $\mathcal{C}_i$, it increments τ by 1, sets $\mathsf{DX}[\ell \mid\mid \tau] := v$, and sends the leakage $\mathcal{L}(\mathsf{DX}, (\mathtt{put}, \ell, v))$ to the simulator Sim.
- $\mathsf{Get}(\ell)$: Upon receiving a label ℓ from the client $\mathcal{C}_i$, it increments τ by 1, sends the leakage $\mathcal{L}(\mathsf{DX}, (\mathtt{get}, \ell, \perp))$ to the simulator Sim, and when it receives a message τ' from the simulator, it returns $\mathsf{DX}[\ell \mid\mid \tau']$ to $\mathcal{C}_i$.

Fig. 3. $F^{\mathcal{L}}_{\mathsf{mKVS}}$: The ideal multi-user KVS functionality parameterized with leakage function $\mathcal{L}$.

associated with τ' to the client. Notice that, unlike single-user ideal functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{KVS}}$, the multi-user ideal functionality can be influenced by the simulator.

Security definition. The real and ideal experiments are the same as in Section 5 with the following differences. First, the experiments are executed not with a single client but with c clients $\mathcal{C}_1 \ldots \mathcal{C}_c$; second, the environment adaptively sends operations to all these clients; and third, the ideal functionality of Figure 1 is replaced with the ideal functionality described in Figure 3.

8.1 Security of the Standard Scheme

We now analyze the security of the standard scheme when its underlying KVS is instantiated with a KVS that does not necessarily satisfy RYW consistency. We start by describing its stateful leakage function.

- $\mathcal{L}(\mathsf{DX}, (\mathsf{op}, \ell, v))$:
  1. if $\mathsf{op} = \mathtt{put}$ output $(\mathtt{put}, \mathsf{opeq}(\ell))$
  2. else if $\mathsf{op} = \mathtt{get}$ output $(\mathtt{get}, \mathsf{opeq}(\ell))$

where opeq is the operation equality pattern which reveals if and when a label was queried or put in the past.

Single-user vs. multi-user leakage. Notice that the leakage profile achieved in the multi-user setting is a function of all the labels whereas the leakage profile achieved in the single-user setting was only a function of the labels that were (exclusively) stored and routed by the corrupted nodes. In particular, this implies that the multi-user leakage is worse than the single-user leakage and equivalent to the leakage achieved by standard (non-distributed) schemes. In following, we will refer to the labels stored and routed exclusively by honest nodes as "honest labels" and to all the other labels as "corrupted labels".

The reason that the single-user leakage is independent of the honest labels is because of the RYW consistency of the underlying KVS. More precisely, RYW

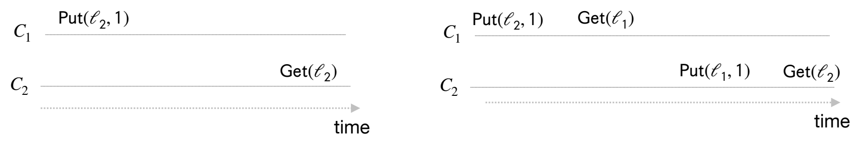

Fig. 4. Sequence 1 is on the left and Sequence 2 is on the right.

consistency guarantees that for a given label, the user will read the latest value that it stored. This implies that the value it reads will be independent of any other label, including the corrupted labels. This is not the case, however, in the multi-user setting where RYW consistency does not guarantee that the honest labels will be independent of the corrupted labels. To see why, consider the following example. Let ℓ_1 be a corrupted label and let ℓ_2 be an honest label. Assume that both ℓ_1 and ℓ_2 initially have the value 0. Now consider the two sequences of operations executed by clients C_1 and C_2 shown in Figure 4. Notice that both sequences are RYW consistent (this is the case because they satisfy a stronger consistency guarantee called *sequential consistency*). However, in sequence 1, $\mathsf{Get}(\ell_2)$ can output both 0 or 1 whereas, in sequence 2, if $\mathsf{Get}(\ell_1)$ outputs a 0, then $\mathsf{Get}(\ell_2)$ can only output 1. This example points out that operations on corrupted labels can impact operations on honest labels. Capturing exactly how operations on one label can effect operations on other labels for different consistency guarantees is challenging but might be helpful in designing solutions with better leakage profiles. We leave this as an open problem. Alternatively, it would be interesting to know if there is some consistency notion one could assume (in the multi-user setting) under which a better leakage profile could be achieved.

Security. We now state our security theorem, the proof of which is in the full version of the paper.

Theorem 4. EKVS *is $\mathcal{L}$-secure with probability at least* $1 - \mathsf{negl}(k)$.

9 Conclusions and Future Work

In this work, we study end-to-end encryption in the context of KVSs. We formalize the security properties of the standard scheme in both the single-user and multi-user settings. We then use our framework to analyze the security of the standard scheme when its underlying KVS is instantiated with consistent hashing based KVS (with zero-hop and multi-hop routing). We see our work as an important step towards designing provably-secure end-to-end encrypted distributed systems like off-chain networks, distributed storage systems, distributed databases and distributed caches.

Our work motivates several open problems and directions for future work.

Relationship between consistency guarantees and leakage. Recall that the standard scheme leaks the operation equality of all the labels in the multi-user setting (with no assumption on the consistency guarantees). However, if the underlying KVS satisfies RYW consistency, the scheme only leaks the operation equality of a subset of labels but in a single-user setting. The most immediate question is whether the leakage can be improved in the multi-user setting by assuming a stronger consistency guarantee.

We however believe that even assuming linearizability, which is much stronger than RYW consistency, the standard scheme would still leak more in the multi-user setting than what it would in the single-user setting with RYW consistency. The question then is to find a lower bound on leakage in the multi-user setting.

Beyond CH-KVS. Another direction is to study the security of the standard EKVS when it is instantiated with a KVS that is not based on consistent hashing or on the two routing schemes that we described. Instantiations based on Kademlia [46] and Koorde [35] would be particularly interesting due to the former's popularity in practice and the latter's theoretical efficiency. Because Koorde uses consistent hashing in its structure (though its routing is different and based on De Bruijn graphs) the bounds we introduce in this work to study CH-KVS's balance might find use in analyzing Koorde. Kademlia, on the other hand, has a very different structure than CH-KVSs so it is likely that new custom techniques and bounds are needed to analyze its balance.

New EKVS constructions. A third direction is to design new EKVS 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 structured encryption and oblivious RAMs.

EKVSs in the transient setting. Another important direction of immediate practical interest is to study the security of EKVSs in the transient setting. As mentioned in Section 4, in transient setting, nodes are not known a-priori and can join and leave at any time. This setting is particularly suited to peer-to-peer networks and permissionless blockchains. Agarwal and Kamara [7] study DHTs in the transient setting and it would be intersting to extend their work to transient KVSs as well.

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 [17]) and adaptive corruptions would be very interesting.

References

1. Apache ignite. https://ignite.apache.org/.
2. Couchbase. https://www.couchbase.com/.

3. Foundationdb. https://www.foundationdb.org/.
4. Memcachedb. https://github.com/LMDB/memcachedb/.
5. Redis. https://redis.io/.
6. Xap. https://www.gigaspaces.com/.
7. A. Agarwal and S. Kamara. Encrypted distributed hash tables. Cryptology ePrint Archive, Report 2019/1126, 2019. https://eprint.iacr.org/2019/1126.
8. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2004.
9. G. Asharov, M. Naor, G. Segev, and I. Shahaf. Searchable symmetric encryption: Optimal locality in linear space via two-dimensional balanced allocations. In *ACM Symposium on Theory of Computing (STOC '16)*, STOC '16, pages 1101–1114, New York, NY, USA, 2016. ACM.
10. G. Asharov, G. Segev, and I. Shahaf. Tight tradeoffs in searchable symmetric encryption. In *Annual International Cryptology Conference*, pages 407–436. Springer, 2018.
11. P. Bailis, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Bolt-on causal consistency. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 761–772, 2013.
12. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *Advances in Cryptology – CRYPTO '07*, Lecture Notes in Computer Science, pages 535–552. Springer, 2007.
13. L. Blackstone, S. Kamara, and T. Moataz. Revisiting leakage abuse attacks. In *Network and Distributed System Security Symposium (NDSS '20)*, 2020.
14. A. Boldyreva, N. Chenette, Y. Lee, and A. O'neill. Order-preserving symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2009*, pages 224–241, 2009.
15. R. Bost. Sophos - forward secure searchable encryption. In *ACM Conference on Computer and Communications Security (CCS '16)*, 20016.
16. 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.
17. R. 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.
18. D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *ACM Conference on Communications and Computer Security (CCS '15)*, pages 668–679. ACM, 2015.
19. D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security Symposium (NDSS '14)*, 2014.
20. 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.
21. D. Cash and S. Tessaro. The locality of searchable symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2014*, 2014.
22. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

23. 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.
24. 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.
25. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS operating systems review*, volume 41, pages 205–220. ACM, 2007.
26. I. Demertzis, D. Papadopoulos, and C. Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In *Advances in Cryptology - CRYPTO '18*, pages 371–406. Springer, 2018.
27. I. Demertzis and C. Papamanthou. Fast searchable encryption with tunable locality. In *ACM International Conference on Management of Data (SIGMOD '17)*, SIGMOD '17, pages 1053–1067, New York, NY, USA, 2017. ACM.
28. M. Etemad, A. Küp**c**cü, C. Papamanthou, and D. Evans. Efficient dynamic searchable encryption with forward privacy. *PoPETs*, 2018(1):5–20, 2018.
29. S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich queries on encrypted data: Beyond exact matches. In *European Symposium on Research in Computer Security (ESORICS '15). Lecture Notes in Computer Science*, volume 9327, pages 123–145, 2015.
30. B. A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, and S. M. Bellovin. Malicious-client security in blind seer: a scalable private dbms. In *IEEE Symposium on Security and Privacy*, pages 395–410. IEEE, 2015.
31. C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC '09)*, pages 169–178. ACM Press, 2009.
32. E.-J. Goh. Secure indexes. Technical Report 2003/216, IACR ePrint Cryptography Archive, 2003. See `http://eprint.iacr.org/2003/216`.
33. O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.
34. M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Network and Distributed System Security Symposium (NDSS '12)*, 2012.
35. M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *International Workshop on Peer-to-Peer Systems*, pages 98–107. Springer, 2003.
36. S. Kamara and T. Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Advances in Cryptology - EUROCRYPT '17*, 2017.
37. S. Kamara and T. Moataz. SQL on Structurally-Encrypted Data. In *Asiacrypt*, 2018.
38. S. Kamara and T. Moataz. Computationally volume-hiding structured encryption. In *Advances in Cryptology - Eurocrypt' 19*, 2019.
39. S. Kamara, T. Moataz, and O. Ohrimenko. Structured encryption and leakae suppression. In *Advances in Cryptology - CRYPTO '18*, 2018.
40. S. Kamara, T. Moataz, S. Zdonik, and Z. Zhao. An optimal relational database encryption scheme. Cryptology ePrint Archive, Report 2020/274, 2020. `https://eprint.iacr.org/2020/274`.
41. S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security (FC '13)*, 2013.

42. S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM Conference on Computer and Communications Security (CCS '12)*. ACM Press, 2012.
43. A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.
44. W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with cops. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 401–416, 2011.
45. W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Stronger semantics for low-latency geo-replicated storage. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 313–328, 2013.
46. P. Maymounkov and D. 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.
47. 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.
48. V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S.-G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind seer: A scalable private dbms. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 359–374. IEEE, 2014.
49. 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.
50. E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *Network and Distributed System Security Symposium (NDSS '14)*, 2014.
51. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.
52. R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. 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.
53. B. Technologies. Riak. `https://docs.basho.com/riak/kv/2.2.2/learn/dynamo/`.
54. Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha. Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 292–308, 2013.