Transparent Dictionaries from Polynomial Commitments

Hossein Hafezi*
New York University

Alireza Shirzad*
University of Pennsylvania

Benedikt Bünz
New York University

Joseph Bonneau
New York University

Abstract

We present IRONDICT, a transparent dictionary construction based on polynomial commitment schemes. Transparent dictionaries enable an untrusted server to maintain a mutable dictionary and provably serve clients lookup queries. A major open challenge is supporting efficient auditing by lightweight clients. Previous solutions either incurred high server costs (limiting throughput) or high client lookup verification costs, hindering them from modern messaging key transparency deployments with billions of users. Our construction makes black-box use of a generic multilinear polynomial commitment scheme and inherits its security notions, i.e. binding and zero-knowledge. We implement our construction with the recent KZH scheme and find that a dictionary with 1 billion entries can be verified on a consumer-grade laptop in 35 ms, a 300× improvement over the state of the art, while also achieving 150,000× smaller proofs (8 kB). In addition, our construction ensures perfect privacy with concretely efficient costs for both the client and the server. We also show fast-forwarding techniques based on incremental verifiable computation (IVC) and checkpoints to enable even faster client auditing. A full version of the paper can be found in [25].

1 Introduction

Many essential services, such as financial systems or messaging platforms, are operated by centralized servers with little accountability. Fully decentralized systems often fall short in practice for performance, cost, privacy or other reasons. As a more pragmatic alternative, some traditionally centralized systems have adopted cryptographic *transparency* to offer increased assurance to the public that they are operating correctly while maintaining their performance benefits. For example, WhatsApp, an encrypted messaging platform with billions of users, rolled out transparency logs for its key-distribution servers in 2024 [34]. Transparency systems ensure all actions by a central server are publicly visible and enable clients to monitor these actions. For example, in the secure messaging context [63], a transparency log ensures clients can observe all public keys the server has advertised for their username (though it cannot prevent the server from advertising maliciously chosen keys). Implicitly, transparency systems assume a *malicious-but-cautious* adversary model [50]. That is, making server misbehavior *detectable* deters misconduct by imposing the risk of reputation damage or legal liability. Transparency has been proposed in a number of contexts, including public-key distribution in encrypted messaging systems [50, 42, 57, 7, 3, 11, 13, 36, 37], distribution of software binaries [20, 26, 2, 44], and certificate authorities via the Certificate Transparency (CT) protocol [53, 23, 35, 31, 32]. All of these applications can utilize a *transparent dictionary*, a cryptographic data structure that supports verifiable queries over a publicly committed dictionary, i.e. a mapping from a set of labels to a set of values. A transparent dictionary typically involves three logical entities:

- The *Server* is a centralized service provider that maintains a mutable dictionary and periodically publishes *commitments* to a public bulletin board. Publication occurs at regular time period or *epochs* (e.g., once every hour). The server also responds to client and auditor queries.
- *Auditors* monitor the bulletin board and ensure that the dictionary is well-formed and maintains promised *global invariants* across epochs.
- *Clients* issue lookup queries to the server to retrieve values associated with labels of interest (e.g. contacts they wish to message). Clients also monitor their own label and may perform auditing of invariants.

Motivating application: key transparency. An important application of transparent dictionaries is *key transparency* [42], in which the server maintains a dictionary, mapping user identities—such as email addresses or usernames—to their public keys. Key transparency promises a practical solution to the age-old challenge of key distribution for encrypted communication systems [63].

It is vital for security that a user can monitor the dictionary at *all* epochs. Otherwise, while a user is offline, a malicious server could insert a malicious public key (enabling impersonation) and later revert the change to avoid detection. This is known as an *oscillation attack* [41]. Hence, key transparency systems should support an efficient mechanism to verify that their public key has remained unchanged throughout multiple epochs during which they were offline. Another important goal is privacy [11]. The server's commitments and proofs should reveal nothing about other users (as an update might indicate a lost device) or system-wide details such as the rate of churn. Note that privacy may not be a concern in other applications, such as software transparency, in which all information is considered public. Scale is another vital concern, as popular platforms are now used by billions of users [55, 19].

1.1 Related Work

Constructions based on Merkle trees. Most transparent dictionary constructions are based on Merkle trees [30, 4, 33, 42, 11, 37, 40]. This enables concretely fast operations for both servers and clients, as the only primitive needed is a collision-resistant hash function (typically SHA-256 or SHA-3). However, a major drawback is that auditors' costs scale linearly with the number of updates per epoch, requiring auditors to perform roughly as much computation as the server itself. As a result, client auditing is infeasible in large deployments. Instead, today's deployments rely on third-party auditors such as non-profit watchdog organizations [56]. At current throughput, auditing WhatsApp's key transparency server requires hashing 200 MB of data every 5 minutes, which is impractical for end users [16].

Constructions from algebraic accumulators. Merkle trees are one construction of a cryptographic *accumulator* or *set commitment*. Transparent dictionaries can also be built from other cryptographic accumulators. Bilinear accumulators [58, 38], for example, offer smaller concrete proof sizes for lookups but are generally impractical due to their high storage and computational overhead. For instance, Tomescu et al. [58] report that supporting a dictionary of size 2^{20} would require hundreds of gigabytes of storage. VeRSA [61] proposed transparent dictionaries based on RSA accumulators [6, 60, 61]. Like our approach, VeRSA supports efficiently verifying batches of updates. VeRSA offers attractive features: its audit procedure is independent of the epoch size, enabling self-auditing by clients. However, generating lookup proofs with VeRSA is concretely expensive and scales linearly with the dictionary size, limiting its applicability to dictionaries of size in the millions (such as binary transparency scenarios), not billions. Many algebraic accumulators also require a setup (possibly trusted) with large public parameters. This drawback also applies to IRONDICT when instantiated with KZH, which requires a linear-sized *structured reference string*. We discuss this further in Sections 3.1 and 5.4.

Reducing auditing costs. Some works [14, 62, 49] have attempted to make client-auditing feasible by delegating auditing to an untrusted party—often the server itself—using SNARKs to prove correctness. In practice, the computational cost of generic SNARK-based designs limits throughput. For example, due to the high cost of SHA-256 in zero-knowledge circuits, Hekaton [49] only achieved a throughput of 10 updates per second on a powerful cluster of 4096 nodes. Verdict [62] took a similar approach but, to reduce circuit size, employed a non-standard zk-friendly hash function [24]. In contrast, our approach reaches significantly higher throughput using just a single server.

Merkle^2 [27] changes the trust model to improve performance. They reduce the auditor's per-epoch workload to logarithmic in the number of key updates by requiring only a single Merkle extension proof per epoch. However, Merkle^2 relies on the *signature chain assumption*, that clients maintain a secure key to sign their own key updates, which we consider unrealistic for secure messaging applications as we will discuss in Section 2.

Table of comparison. Figure 7 compares the asymptotic costs of IRONDICT with prior work. The key advantage of IRONDICT lies in its audit and update costs, which remain independent of the epoch size. As we will later show in Section 5, IRONDICT is concretely efficient since its main operations are field operations, which are concretely efficient.

1.2 Our Contributions

- **A simple transparent dictionary based on polynomial commitments.** Our main contribution is IRONDICT, a novel transparent dictionary built upon a generic polynomial commitment scheme (PCS) defined over a large field[1], a new paradigm in designing transparent dictionaries. Conceptually, our design is simple, with most of the complexity encapsulated within the underlying PCS scheme. The construction provides (I) low server overhead resulting in a high throughput and (II) low auditing overhead, with the latter being independent of the update batch size. Instantiating IRONDICT with KZH as the underlying PCS [28] yields a concretely efficient implementation that enables lightweight auditing: audit proofs for a dictionary supporting 1 billion users are under 8 kB and verifiable in about 35 ms on a personal computer, while sustaining a throughput of more than 1,000 updates (or user enrollments) per second.
- **Doubly-efficient consistency proofs.** We introduce a new approach for verifying *consistency* in transparent dictionar-

[1] We require the PCS to be defined over a field $\mathbb{F}$ with sufficiently large cardinality—typically $|\mathbb{F}| \geq 2^{128}$ or 2^{256}—to ensure that birthday attacks on a hash function $H : \{0,1\}^* \to \mathbb{F}$ are computationally infeasible.

ies; that is, a proof in the common case that an individual label-value mapping was unchanged over many epochs. Unlike prior work, our method eliminates the need to maintain explicit value histories or versioning. We show that consistency can be checked using just two PCS openings (or even a single one if the underlying PCS supports homomorphism), yielding a highly efficient solution for both the server and the client. Concretely, the server can produce a consistency proof with size 5 kB in under 35 ms, while the client can verify it in under 30 ms.

- **Concretely efficient fast-forwarding for auditors via IVC.** Fast-forwarding is the ability for a new auditor—or an auditor who has been offline for n epochs—to verify the current state by checking only a sublinear number of proofs, rather than auditing all n epochs individually. We introduce an IVC [64] scheme for fast-forwarding auditors for IRONDICT, such that the IVC cost is independent of the size of the update batch. When IRONDICT is concretely instantiated with KZH, we estimate to achieve $1000\times$ prover performance improvement in practice compared to VeRSA [61], which also proposed using an IVC scheme with constant circuit size per step. Due to the lack of space, we defer its details to the full version of the paper.
- **Checkpoints for efficient self-auditing by clients.** Inspired by VeRSA, we propose an alternative to IVC that allows clients to fast-forward through updates using lightweight checkpoints. When a client comes online, it only requires verifying a short (sublinear-length) sequence of checkpoint proofs spanning from its last known state to the current one. This approach enables efficient self-auditing without incurring the computational overhead associated with IVC-based methods. Due to the lack of space, we defer the details of our checkpoints approach to the full version of the paper.
- **A stronger definition of privacy.** Prior works define privacy in transparent dictionaries by requiring (I) the commitment to the dictionary to be hiding (II) the client queries and audit proofs reveal no more information than what is permitted by a predefined *leakage function*. These leakage functions are typically tailored to specific constructions and do not offer strong cryptographic guarantees. For example, in SEEMless [11], lookup queries additionally reveal the last epoch during which a value was updated—a non-standard and unintended leakage arising from their design. In this work, we propose zk-IRONDICT with a strong, zero-knowledge-based notion of privacy: clients and auditors learn *nothing* beyond the correctness of their queries and (an upper bound on) the dictionary's size. Our definition is generic and applies to any PCS that satisfies the hiding property and supports zero-knowledge openings [9].
- **A zero-knowledge variant of KZH.** To meet the privacy requirements, the underlying PCS must have a hiding commitment and a zero-knowledge opening protocol. KZH lacks these properties natively. Prior work [9] proposes a generic compiler that transforms a homomorphic PCS to one that has a zero-knowledge opening protocol. There are two downsides to this approach: (I) it is expensive, and (II) it lacks a formal proof of the zero-knowledge property. In this work, we formally prove the security of this compiler and show that, when applied to KZH, only a sublinear amount of randomness suffices, which makes the compiler lightweight. This contribution can be of independent interest.
- **Memory optimizations.** By taking advantage of the homomorphic properties of KZH, when IRONDICT is initiated with KZH, we introduce a trade-off between computation and memory. Specifically, at each epoch, the server only needs to store the difference between the current and previous epoch, reducing the memory overhead. However, when the server needs to *open* a value later, it incurs a computational cost, performing a number of field operations proportional to the size of the epoch. Due to the lack of space, we defer the details of our memory optimization to the full version of the paper.
- **Implementation.** We implement IRONDICT and demonstrate that it scales to dictionaries with billions of entries, matching the demands of modern secure messaging applications, while supporting update rates of up to thousands per second. We conduct our experiments on a Google `c4-highmem-144-lssd` instance, equipped with 144 CPU cores and 1116 GB of memory. For a dictionary supporting 1 billion users, our system achieves an epoch interval of nearly 15 minutes while sustaining a throughput of over 1000 updates per second. Auditor proofs remain under 8 kB and can be verified in under 35 ms on a consumer-grade laptop. Client queries—including lookups and consistency checks—produce proofs of approximately 6 kB, which can be verified in less than 30 ms.

2 System model

We outline the basic roles in our system here, which is consistent with prior work on key transparency [42]:

Server. The main entity in our system is a *server* that provides access to a queryable dictionary (a label-value map) and periodically updates it (once per *epoch*) while maintaining some invariants. With each update, the server publishes a new commitment to the dictionary. We expect the server to provide verifiable answers to two types of queries:

- Lookup: What value is assigned to label ℓ in epoch i?
- Consistency: Has the value assigned to the label ℓ changed between epochs i and j?

Note that simply performing lookups on the values mapped to label ℓ at epochs i and j does not suffice as a consistency

proof between i and j. The server might attempt an *oscillation attack* [41] by changing the mapping for a targeted label at one epoch and then switching it back in the next epoch. Hence, a consistency proof must rule out changes in any epoch between i and j. In some systems, the server commits to maintaining additional invariants. For example, a version invariant [42, 7] attaches a version number v to each value and requires that this is incremented in any epoch in which the value changes. Given this invariant, checking that a value has the same version number in epochs i and j is equivalent to a consistency proof.

Client. A set of *clients* (or users) are able to query the dictionary via the server. Typically, each client is represented by a label ℓ, which is a human-readable identifier such as a username, email address, or phone number.[2] The client will typically *monitor* this label by requesting consistency proofs between epochs and manually verifying any changes to the value mapped to their label. Clients will also request lookup proofs of other labels of interest, such as peers in a secure messaging system that they wish to communicate with. However, the clients *do not* monitor other labels over time. The assumption is that the label's owner will monitor and detect any unauthorized changes. Clients can also request updates to the value mapped to their label (for example, to change their public key after losing a device). Notably, we do not assume any cryptographic authentication (such as a signature) is required to update any mapping, as some prior work does [27]. The downside of such assumptions is that they require users to maintain uninterrupted access to cryptographic keys over time. In practice, users often lose their devices or reinstall applications, thereby losing access to previous secret keys. Recovering from such scenarios inherently requires the server to update any label's value based on non-cryptographic authentication (e.g. passwords or 2FA). This has important implications for monitoring in that surreptitious updates by the server are *detectable* but not *provable*.

Auditors. *Auditors* monitor the server's updates and verify the correctness of a proof of invariance at each epoch. At a high level, this proof ensures that the transition from one epoch to the next preserves the desired invariants. For example, under the versioning invariant described previously, a valid transition requires that if a value is modified, its version number must also be incremented accordingly. Importantly, the auditors only check for the server's *detectable* and *provable* misbehaviour, such as an incorrect proof of invariance. Some *whistle-blowing procedure* must be in place to broadcast evidence of misconduct, which we leave out of scope. The efficiency of auditing depends directly on the size and verification cost of invariance proofs. For example, consider the simple case where the server publishes all value updates directly on the bulletin board. In this setting, auditors must be *almost* as powerful as the server to recompute the updates and verify their correctness. However, the assumption of such powerful auditors means most clients will be incapable of auditing and must rely on a quorum of third parties to audit. While any one correct auditor can broadcast proofs of misbehaviour, the incentives for performing expensive auditing remain unclear. Instead, we aim to make *self-auditing* possible, where even lightweight clients are capable of performing auditing.

Bulletin board. Like prior work, we assume a public bulletin board, e.g. a publicly accessible and append-only ledger that allows parties to publish arbitrary strings along with authentication tags for verification. We treat the bulletin board as a black box with the following basic functionality: (I) post operation: A party can publish a string on the bulletin board, and (II) retrieve operation: Anyone can query the bulletin board. The use of a bulletin board prevents split-view attacks, where a malicious server provides different clients with inconsistent states of the dictionary at the same epoch i. The bulletin board abstraction can be realised in various ways, e.g. a public blockchain [59] or a gossip protocol [54, 41]. We leave the details out of scope, as they do not affect the design of our dictionary. We denote the public bulletin board via BB.

3 Preliminaries and building blocks

Notation. We denote vectors using bold letters, such as $\mathbf{w}$, and use w_i to refer to the i-th entry of the vector. Let $\mathbb{F}$ be a finite field and $\mathbb{G}$ a group with scalars in $\mathbb{F}$, with additive notation. For $a \in \mathbb{F}$ and $\mathsf{G} \in \mathbb{G}$, the scalar multiplication of G by a is denoted as $a \times \mathsf{G}$. A function $f(x)$ is negligible if, for any polynomial $p(x)$, there exists a positive integer N such that for all $x > N$, we have $f(x) < \frac{1}{p(x)}$. When we say an event happens with overwhelming probability, we mean it occurs with a probability of $1 - \varepsilon(\lambda)$, where $\varepsilon(\lambda)$ is a negligible function.

Dictionary. We denote by a dictionary Dict as a set of label-value pairs $\{(\mathsf{label}_i, \mathsf{value}_i)\}_{i=1}^{m}$ with *unique* label values. For a dictionary Dict, we denote by $\mathsf{Dict}[\mathsf{label}]$ the value corresponding to label in the dictionary Dict; i.e.,

$$\mathsf{Dict}[\mathsf{label}] := \begin{cases} \mathsf{value} & \text{if } (\mathsf{label}, \mathsf{value}) \in \mathsf{Dict} \\ \perp & \text{otherwise.} \end{cases}$$

When we refer to a dictionary as $\mathsf{Dict} : A \to B$, it means its labels are in set A and its values are in set B.

Multilinear polynomials. A multivariate polynomial is said to be *multilinear* if the individual degree of any variable X_i is at most 1. We use $\mathbb{F}_{\mu}^{\leq 1}[\mathbf{X}]$ to denote the space of all μ-variate multilinear polynomials. A multilinear polynomial $\overline{p} \in \mathbb{F}_{\mu}^{\leq 1}[\mathbf{X}]$ of size $N = 2^{\mu}$ is uniquely defined by (or uniquely extends) its N evaluations on the Boolean hypercube. We

[2] Later, we distinguish between human-readable labels and *indices* which are assigned to each user but not exposed at the UX level.

denote by $\mathsf{L}_{\mathbf{w}}(\mathbf{X})$ a multilinear Lagrange polynomial that vanishes all over the Boolean hypercube $\{0,1\}^{\mu}$ except on $\mathbf{w} \in \{0,1\}^{\mu}$. It can be computed as follows:

$$\mathsf{L}_{\mathbf{w}}(\mathbf{X}) = \prod_{i=1}^{\mu}(X_i w_i + (1-X_i)(1-w_i))$$

Multilinear polynomial commitment scheme (PCS). PCS is a cryptographic primitive that enables a prover to *commit* to a multilinear polynomial succinctly and later prove the correctness of its evaluation at some chosen points from the domain, without revealing the full polynomial. When a verifier requests an opening at a point $\vec{x}$, the prover returns both the claimed evaluation $y = p(\vec{x})$ and a proof π attesting to its correctness. The verifier can efficiently check that y is the correct value of the committed polynomial at $\vec{x}$ using the proof π. A formal definition of PCS and its required properties is deferred to the full version of the paper.

Zerocheck. As a core primitive, we use *zerochecks* (see [12] for a formal definition), which is a protocol to show that a low-degree composition of committed multilinear polynomials evaluates to 0 on $\{0,1\}^{\mu} \subset \mathbb{F}^{\mu}$. Zerochecks reduce to a sumcheck protocol [39] and a PCS opening.

3.1 KZH: PCS with optimal opening

In this paper, we choose KZH [28] to instantiate the PCS for our construction. KZH is a family of multilinear PCS built on a pairing-friendly elliptic curve. A key feature of KZH is that it has a sublinear opening proof size and a sublinear opening time for both the prover[3] and the verifier, similar to Hyrax [65]. However, unlike Hyrax and similar to KZG [29], KZH has a commitment of constant size, consisting of a single group element. KZH (more precisely KZH-2) has an opening size and verifier time of $O(N^{\frac{1}{2}})$ and is generalized to a family of schemes called KZH-k for $k \geq 2$, which achieves an opening size and verifier time of $O(k \cdot N^{\frac{1}{k}})$ for a constant k. Due to lack of space, we defer a full description of KZH-k as well as a table comparison between different KZH variants to the full version of the paper. In this paper, when we mention KZH, we refer to the entire KZH-k family. If we are specifically discussing one, we specify it with the index k. KZH is secure in the AGM under the (q_1, q_2)-*dlog* and *setup-find-rep* assumptions.

Advantages of KZH for our scheme. Apart from sublinear proof and verifier time, the KZH family has two key advantages that make it particularly well-suited as the underlying PCS for our construction:

- *Free opening at Boolean points.* Opening Boolean points using KZH-k is essentially *free*, meaning the server is only required to read k *arrays* of size $N^{1/k}$ from memory. The term *free* here refers to the fact that the server does not need to perform any computation to generate the proof; rather, it only needs to read and return the relevant auxiliary data. This is analogous to Merkle trees, where the opening of a leaf involves no computation and consists solely of reading and returning internal nodes. As we will see in the next section, the server only needs to open the polynomial at Boolean points in response to client requests. Therefore, the efficiency of openings at Boolean points—compared to non-Boolean points—is especially important in our setting.

- *Homomorphic commitments and auxiliary input.* KZH family is homomorphic, and its auxiliary input used for opening is homomorphic as well. This means that updating n evaluation points of a polynomial only requires $k \cdot n$ group operations to update both the commitment and the auxiliary data of KZH-k. Although our scheme does not necessarily require homomorphism, homomorphism significantly improves the efficiency of our protocol for value consistency.

KZH setup. One limitation of KZH is its requirement for a *linear-sized* SRS, similar to the trusted setup used in the KZG scheme based on powers-of-tau. The SRS must be generated via a *trusted setup*, which introduces important security considerations. In practice, such setups are typically carried out using multi-party computation (MPC) ceremonies, where multiple participants sequentially contribute randomness to the SRS. As long as *at least one participant acts honestly*, the resulting SRS remains secure. This approach has been successfully employed by real-world systems such as Zcash [8] and Filecoin [46]. A summary of current best practices for MPC ceremonies is provided in [66]. Concrete numbers of the KZH setup are provided in Section 5.4.

3.2 Transparent dictionary (TD)

Definition 1 (Transparent dictionary)**.** A *transparent dictionary protocol* is defined with respect to three parties: Server, Auditor, and Client, all of whom have access to a public bulletin board BB. Because of the existence of BB, we assume all parties know the current epoch number i. Each party supports the following algorithms:

- $\mathsf{Setup}(\lambda, \mu) \to (\mathsf{key}_{\mathsf{Server}}; \mathsf{key}_{\mathsf{Auditor}}, \mathsf{key}_{\mathsf{Client}})$: Given a security parameter λ and log of maximum supported size μ, this algorithm respectively outputs participants' keys, namely: $\mathsf{key}_{\mathsf{Server}}$, $\mathsf{key}_{\mathsf{Auditor}}$ and $\mathsf{key}_{\mathsf{Client}}$. We assume algorithms corresponding to each party take their corresponding key as an implicit input; we omit it for brevity. If the setup requires secret randomness, we assume it is executed by a trusted party (e.g., implemented via a ceremony).

- $\mathsf{Server.Init}() \to (\mathsf{state}_0, \mathsf{com}_0)$: Initializes the dictionary state as state_0 and computes a succinct commitment com_0

[3] Excluding the polynomial evaluation itself.

to this state. The commitment com_0 is then published on the bulletin board BB.

- $\mathsf{Server.Update}(\mathsf{state}_{i-1}, \Delta) \to (\mathsf{state}_i, \mathsf{com}_i, \pi_i)$: Updates the current state state_{i-1} using a set of changes Δ to obtain a new state state_i. It then computes and publishes a new commitment com_i to BB, along with invariance proof π_i.

- $\mathsf{Server.Lookup}(\ell, \mathsf{state}_i) \to (\mathsf{value}, \pi_{\mathsf{Lookup}})$: Given a label ℓ and epoch state state_i, returns the value value associated with ℓ in state state_i, along with a lookup proof π_{Lookup}. If state_i does not exist or ℓ is invalid, the algorithm returns $\perp$.

- $\mathsf{Client.VerifyLookup}(\ell, \mathsf{com}_i, \mathsf{value}, \pi_{\mathsf{Lookup}}) \to 0/1$: Given a label ℓ, epoch commitment com_i, value value, and a lookup proof π_{Lookup}, this algorithm verifies the lookup and returns 1 (accept) or 0 (reject).

- $\mathsf{Auditor.VerifyInvariance}(\mathsf{com}_i, \mathsf{com}_{i+1}, \pi_{i+1}) \to 0/1$: Given commitments com_i and com_{i+1} for epoches i and $i+1$ along with proof of invariance π_{i+1}, verifies its validity and returns 1 (accept) or 0 (reject).

- $\mathsf{Server.Consistency}(\ell, \mathsf{state}_i, \mathsf{state}_j) \to \pi_{\text{consis}}$: Given a label ℓ and two states state_i and state_j corresponding to epochs i and j with $i < j$, the server returns a proof π_{consis} attesting that the value associated with ℓ remained unchanged throughout epochs i to j. If the value changed during this interval, it returns $\perp$.

- $\mathsf{Client.VerConsistency}(\ell, \mathsf{com}_i, \mathsf{com}_j, \pi_{\text{consis}}) \to 0/1$: Client verifies validity of a consistency proof π_{consis} for the label ℓ between epochs i and j, returning 1 if the proof is accepted, and 0 otherwise.

Comparison to [11]. Our definition adapts that of [11] with several refinements. In particular, we introduce an explicit Setup algorithm that outputs keys for the participating parties. Unlike Server.Init, which is typically executed by the server and may be deterministic, Setup can involve secret randomness and is not necessarily run by the server. Finally, we simplify history-related functions by focusing solely on consistency, replacing full change histories with a functionality that checks if a value has changed within a given epoch range, making the definition more simplified for client applications.

Completeness and Soundness. Completeness ensures that when the setup is performed honestly and the server behaves correctly, the system behaves as expected. In particular, consistency correctness guarantees that if a client requests a value consistency check between epochs i and j for a value that has not changed, then running Client.VerConsistency on the proof will lead to acceptance, and any query for the value within the interval $[i, j)$ returns the same result. Similarly, audit correctness ensures that at each epoch, any honest auditor executing Auditor.VerifyInvariance outputs 1.

Soundness, on the other hand, ensures security except with negligible probability. Lookup soundness (binding) guarantees that given a commitment com_i for an epoch and a label ℓ, it is computationally infeasible for the server to produce two distinct values with valid proofs that both pass verification under Client.VerifyLookup for the same commitment. Consistency soundness ensures that if at least one honest auditor accepts the proof for each epoch, then for any two epochs i and j, the server cannot generate a proof of consistency for a value that changes during the interval $[i, j]$ that will be accepted by the client. We defer formal definitions of completeness and soundness to the full version of the paper.

4 IRONDICT

In this section, we describe our construction, which is based on a generic PCS. In Section 4.1, we introduce how a restricted class of dictionaries can be modeled using a single multilinear polynomial, and we explain why this approach fails for general string-to-string dictionaries. In Section 4.2, we extend the previous idea and show how to construct general dictionaries—mapping arbitrary strings to arbitrary strings—using two multilinear polynomials, referred to as Index and Value polynomials. We also specify the required properties of each polynomial. Section 4.3 focuses on Index polynomial and its *append-only* property, discussing associated design trade-offs. Then, in Section 4.4, we present how Value polynomial is used to support *value consistency* proofs. Finally, in Section 4.5, we describe the complete construction using a generic PCS. In Section 4.6, we describe our notion of privacy and introduce our privacy-preserving variant zk-IRONDICT. Finally, we provide a cost overview of the construction instantiated with the KZH family. This includes the additional costs incurred to make the scheme privacy-preserving—that is, to transform KZH zero-knowledge.

4.1 Polynomial Encodings for Dictionaries

Modeling a restricted class of dictionaries. Given two natural numbers m and N, such that we assume N is a power of two, consider a dictionary represented as a set of pairs of labels index_i and values value_i:

$$\{(\mathsf{index}_i, \mathsf{value}_i)\}_{i\in[m]} : \begin{cases} \mathsf{index}_i \in \{0,1\}^{\log_2 N} \\ \mathsf{value}_i \in \mathbb{F} \setminus \{0\} \end{cases}$$

We model this dictionary as a multilinear polynomial $\overline{p} \in \mathbb{F}^{\leq 1}_{\log_2 N}$, where $\overline{p}(\mathsf{index}_i) = \mathsf{value}_i$ for each stored index-value pair, and $\overline{p}(\mathbf{x}) = 0$ for all indices $\mathbf{x}$ that do not appear in the dictionary, in other words:

$$\overline{p}(\mathbf{X}) = \begin{cases} \mathsf{value}_i & : \mathbf{X} = \mathsf{index}_i \\ 0 & : \mathbf{X} \in \{0,1\}^{\log_2 N} \setminus \{\mathsf{index}_i\}_{i\in[m]} \end{cases}$$

We are implicitly using the fact that given multilinear polynomial evaluations over the Boolean hypercube, it determines a single unique polynomial.

Modeling a general dictionary. The dictionary introduced earlier represents a restricted case, where non-zero field elements are mapped to fixed-length binary strings $\{0,1\}^{\log_2 N}$. In contrast, our goal is to support a more general type of dictionary that maps arbitrary strings to arbitrary strings. To enable this, we follow the standard approach of hashing arbitrary inputs into a fixed domain. Specifically, we employ a collision-resistant and one-way hash function $H : \{0,1\}^* \to \mathbb{F}$. Provided that $\mathbb{F}$ is sufficiently large to prevent birthday attacks, H can be treated as an injective map from $\{0,1\}^*$ to $\mathbb{F} \setminus \{0\}$. However, the same argument does not apply to mappings into $\{0,1\}^{\log N}$, since avoiding birthday attacks requires the size of $\{0,1\}^{\log N}$ to exceed 2^{128}. Standard PCS constructions are not well-suited for such sparse polynomials, as even a square-root-sized SRS would be prohibitively large. Sparse-to-dense techniques such as Spark [52], while applicable, incur inefficient openings due to their reliance on knowledge arguments. To overcome these challenges, we model the dictionary using two dense polynomials rather than a single sparse one..

4.2 Modeling general dictionaries

Modeling dictionaries. To model a dictionary with domain $\mathsf{Dict} : \{0,1\}^* \to \{0,1\}^*$, we build it by composing two different dictionaries Index and Value as described:

$$\mathsf{Index} : \mathbb{F} \setminus \{0\} \to \{0,1\}^{\log_2 N}, \mathsf{Value} : \{0,1\}^{\log_2 N} \to \mathbb{F} \setminus \{0\}$$

which, by composability, implies that:

$$\mathsf{Dict} = (\mathsf{Index}, \mathsf{Value}) : \mathbb{F} \setminus \{0\} \to \mathbb{F} \setminus \{0\}$$

To retrieve the value associated with label from $\mathsf{Dict} = (\mathsf{Index}, \mathsf{Value})$, the user first looks up the corresponding index in Index, then uses this index to query Value. Here is a quick overview of the Index and Value dictionaries (polynomials). We'll explore them in depth in the next subsections.

Index dictionary. An append-only dictionary, denoted as $\mathsf{Index} : \mathbb{F} \setminus \{0\} \to \{0,1\}^{\log_2 N}$, we exclusively use it for assigning a fixed point on the Boolean hypercube to a label. This dictionary maps labels (e.g., email addresses) to a *unique index* within $\{0,1\}^{\log_2 N}$, where N represents the system's capacity, i.e., the maximum number of values it can support. When adding a new label, the server must assign the label a fresh index. The server does so by applying *open addressing* [67]. Note that the uniqueness of indexes implies that their inverse dictionary is also a dictionary.[4]

[4] In fact, we model its inverse, $\mathsf{Index}^{-1} : \{0,1\}^{\log_2 N} \to \mathbb{F} \setminus \{0\}$, using a multilinear polynomial. However, since the dictionary is invertible, this distinction does not matter.

Value dictionary. A dictionary $\mathsf{Value} : \{0,1\}^{\log_2 N} \to \mathbb{F} \setminus \{0\}$ that maps user indices to their values. Unlike Index, which is append-only and maintains user indices, this dictionary allows updates to values. Maintaining two separate dictionaries allows the system to accommodate different update rates in practice since their updating procedure is different, as we will see—for instance, values might be updated more frequently than new users are registered (assigned new indexes). We construct a transparent dictionary Dict with a pair of multilinear polynomials encoding the dictionaries Index and Value separately; namely $\overline{\mathsf{index}}$ and $\overline{\mathsf{value}}$. We refer to the indexed version of these polynomials, $(\overline{\mathsf{index}}_i, \overline{\mathsf{value}}_i)$, as the state of the dictionary at epoch i. The server uses a PCS to commit to these polynomials and publish the commitments on the bulletin board BB.

4.3 Index dictionary

As previously mentioned, the server must prove the correctness of the index dictionary at every step to prevent attacks that exploit inconsistent indexing. For each user, the client must accept a single, unique index on the Boolean hypercube $\{0,1\}^{\log_2 N}$. If the server provides multiple indexes corresponding to the same label ℓ, a split-view attack can occur. Specifically, consider the case where the server assigns two different indexes $\mathbf{y}_1, \mathbf{y}_2 \in \{0,1\}^{\log_2 N}$ to the same label ℓ. If these indexes produce different evaluations under the $\overline{\mathsf{value}}$ function, i.e., $\overline{\mathsf{value}}(\mathbf{y}_1) \neq \overline{\mathsf{value}}(\mathbf{y}_2)$, then the server can take advantage of this mapping to present different values to different clients. Ensuring correctness means proving that each label is consistently assigned to a single index in all interactions. To assign indices to other labels, we employ an open addressing strategy inspired by sparse Merkle trees [18]. In addition to the hash function $H : \{0,1\}^* \to \mathbb{F}$, we also introduce another hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{\log_2 N}$. The server assigns an index to a label ℓ using the algorithm described in Figure 1. The server is only allowed to assign one of the indexes $\mathcal{H}(0,\ell), \mathcal{H}(1,\ell), \ldots$ to the user in sequential order. Any deviation from this order renders the index invalid. Now, suppose that the server assigns $\mathcal{H}(m_0,\ell)$ to the user. When the server opens the index to the client, it must also prove that all previous indexes $\mathcal{H}(m,\ell)$ for $m < m_0$ were occupied and are not equal to $H(\ell)$. Note that the server might arbitrarily set $\overline{\mathsf{index}}(\mathbf{y}) = H(\ell)$ for some point $\mathbf{y}$. However, since the client requires an identifier $m_0 \in \mathbb{N}$ such that $\mathbf{y} = \mathcal{H}(m_0,\ell)$, it would not affect the client. The verification procedure is described in Figure 2.

Epoch consistency proof. We require Index dictionary to be append-only, meaning that all indexes in the previous state must be included in the current state. Assume $\overline{\mathsf{index}}_i$ and $\overline{\mathsf{index}}_{i+1}$ are the states of the index dictionary in epochs i and $i+1$. The server must prove that: $\overline{\mathsf{index}}_i \subseteq \overline{\mathsf{index}}_{i+1}$, which means that the new dictionary Index_{i+1} includes all the previous indexes from Index_i. This condition is expressed as,

(1) Set $m := 0$, now given label ℓ, the server computes $\mathbf{y} := \mathcal{H}(m, \ell)$ and checks if the index $\mathbf{y}$ is not already assigned, in other words, $\overline{\text{index}}(\mathbf{y}) = 0$. If so, it sets $\overline{\text{index}}(\mathbf{y}) = H(\ell)$ and returns $\mathbf{y}$ as the assigned index.
(2) If $\overline{\text{index}}(\mathbf{y}) \neq 0$, the server updates $m := m + 1$ and then recompute $\mathbf{y} := \mathcal{H}(m, \ell)$ and repeats the procedure. This continues until an unassigned index is found.

Figure 1: Index assignment procedure by the server

(1) The client wishes to verify that a given index $\mathbf{y}$ corresponds to a label ℓ.
(2) The client receives an identifier $m_0 \in \mathbb{N}$ and the index proof, which consists of the corresponding PCS openings of $\overline{\text{index}}$ at points $\mathbf{y}_m := \mathcal{H}(m, \ell)$ for all $m \leq m_0$.
(3) The client runs PCS verification to validate the correctness of PCS openings.
(4) The client for $m < m_0$, checks that $\overline{\text{index}}(\mathbf{y}_m) \neq 0$, $\overline{\text{index}}(\mathbf{y}_m) \neq H(\ell)$. Finally, it verifies that $\mathbf{y} = \mathbf{y}_{m_0}$ and $\overline{\text{index}}(\mathbf{y}) = H(\ell)$.

Figure 2: Verification of index assignment by the client

for all $\mathbf{x} \in \{0,1\}^{\log_2 N}$:

$$Z(\mathbf{x}) := \overline{\text{index}}_i(\mathbf{x}) \cdot \left(\overline{\text{index}}_{i+1}(\mathbf{x}) - \overline{\text{index}}_i(\mathbf{x})\right) = 0$$

$$\implies \sum_{\mathbf{w} \in \{0,1\}^{\log_2 N}} \mathsf{L}_{\mathbf{w}}(\mathbf{X}) \cdot Z(\mathbf{w}) = 0$$

The condition above corresponds to a zerocheck [12]. At a high level, it requires the server to publish a sumcheck proof along with openings of the polynomials $\overline{\text{index}}_i$ and $\overline{\text{index}}_{i+1}$ at a random challenge point $\mathbf{r}$. Crucially, since both polynomials are opened at the same point, a homomorphic PCS enables these openings to be *batched* into a single one by verifying a random linear combination of the two commitments, thereby reducing the verification cost and the communication on the bulletin board. Since the sumcheck proof is relatively inexpensive in practice, the auditor's cost is primarily dominated by the PCS verification(s), e.g. a multi-pairing of size $k \cdot N^{1/k}$ when using KZH-k, where N denotes the size of the dictionary.

Overhead of index proof. One potential issue with this approach arises when the identifier m_0 is large. In such cases, the server must open m_0 points on $\overline{\text{index}}_i$, increasing both communication and computation for the server and client. To address this, we *overprovision the dictionary capacity*. This over-provisioning is necessary because labels are mapped to indexes using a random mapping, e.g. a hash fucntion. As the number of labels approaches the dictionary's capacity N, the probability of hash collisions increases. By setting the nominal capacity to be $2 \leq \alpha$ times the expected number of labels, we can bound the probability of a collision to $\frac{1}{\alpha}$. The expected number of openings can consequently be bounded by $\frac{\alpha-1}{\alpha}$, e.g. ≈ 1.33 for $\alpha = 4$. α increases the SRS size by a factor of α and the verifier time by $\alpha^{1/k}$ but interestingly hardly effects the prover time. This is because committing to zeros is essentially free in Pedersen-like commitment schemes, like KZH.

Worst-case analysis. Similar to most prior works [42, 11, 37], we employ an asymmetric, keyed Verifiable Random Function (VRF) [43] instead of a simple hash function $\mathcal{H}$ to assign indices to users. This choice enhances both security and privacy because the output of a hash function is deterministic and publicly computable, allowing adversaries to precompute hashes offline and potentially link dictionary entries to known labels—enabling various attacks [42]. Specifically, if $\mathcal{H}$ were directly evaluable by an adversary, they could craft labels that produce large identifiers m_0, increasing the computational burden of the server. In contrast, the output of a VRF cannot be computed by clients themselves, effectively preventing brute-force attempts. As a result, we can assume that the output of $\mathcal{H}$ is uniformly random and has not been precomputed by the adversary. This allows our worst-case analysis to align closely with the average-case scenario.

Comparison to cuckoo hashing. A related strategy to open addressing for handling collisions in a hash table is cuckoo hashing [45, 22]. In open addressing, when a collision occurs, the algorithm searches for the next available slot, but already-inserted elements remain untouched. In contrast, cuckoo hashing uses multiple hash functions and allows each key to occupy one of several candidate positions. When a collision occurs, the new key displaces the existing key, which must then be reinserted into the table, potentially triggering a chain of displacements. While cuckoo hashing can more efficiently utilize available slots, open addressing is a more natural fit for our purposes as previously inserted elements are never displaced. This lets us take advantage of the append-only property, which can be naturally and simply expressed using polynomial equations. We leave an investigation of cuckoo hashing-based index polynomial to future work.

4.4 Value consistency

An important feature of a transparent dictionary is *value consistency*, a resource-constrained client should be able to succinctly verify that its value has not changed over a specific period. This translates to checking whether the value corresponding to a specific label has remained unchanged from epoch i to epoch j. A naive approach would involve opening all commitments $\overline{\text{value}}_i$, $\overline{\text{value}}_{i+1}$, …, $\overline{\text{value}}_j$, at the index corresponding to the client and verifying that the value has not changed, resulting in linear number of lookups. Our solution leverages randomness and is concretely more efficient when using a homomorphic PCS; we refer to this approach

as *RLC (random linear combination) based approach*. In the RLC-based approach, when the PCS is homomorphic such as KZH, the server performs $O(n)$ group operations per epoch, where n is the number of updated entries. This yields constant-sized communication on BB and constant auditor verification time. The server simply publishes a few PCS commitments on BB, and the auditor checks a homomorphic relation between them, i.e. equation (1). However, if the PCS is not homomorphic, non-homomorphically evaluating the same check requires four polynomial openings—due to the four distinct polynomials in equation (1)—and publishing their openings at a random point on BB.

RLC-based approach. We define the difference polynomial $\overline{\Delta_i}(\mathbf{X})$, which is essentially the polynomial corresponding to the update at epoch i, as below:

$$\overline{\Delta_i}(\mathbf{X}) = \overline{\text{value}}_i(\mathbf{X}) - \overline{\text{value}}_{i-1}(\mathbf{X})$$

$$\implies \overline{\text{value}}_j(\mathbf{X}) = \overline{\text{value}}_i(\mathbf{X}) + \sum_{t=i+1}^{j} \overline{\Delta_t}(\mathbf{X})$$

The naive approach to verifying whether a value remains unchanged between two epochs i and j is to simply open the value at both epochs and compare the values for equality. However, this method is vulnerable to a *ghost-value attack*, in which the server temporarily modifies the value during some intermediate epoch and later reverts it to its original value. To prevent such attacks, we introduce randomness in a way that ensures changes to a value–even if reverted–cannot cancel each other out. Consider the following polynomial, which is recursively defined as follows:

$$\overline{\text{rand}}_i(\mathbf{X}) = \overline{\text{value}}_0(\mathbf{X}) + \sum_{t=1}^{i} r_t \cdot \overline{\Delta_t}(\mathbf{X}) = \overline{\text{rand}}_{i-1}(\mathbf{X}) + r_i \cdot \overline{\Delta_i}(\mathbf{X})$$

At each epoch, the server publishes a commitment to $\overline{\text{rand}}_i(\mathbf{X})$, where correctness can be verified by auditors using the homomorphism through the equation

$$\begin{aligned}\overline{\text{rand}}_i(\mathbf{X}) &= \overline{\text{rand}}_{i-1}(\mathbf{X}) + r_i \cdot \overline{\Delta_i}(\mathbf{X}) \\ &= \overline{\text{rand}}_{i-1}(\mathbf{X}) + r_i \cdot \left(\overline{\text{value}}_i(\mathbf{X}) - \overline{\text{value}}_{i-1}(\mathbf{X})\right)\end{aligned} \tag{1}$$

The randomness can be derived via the Fiat–Shamir heuristic [21], with an auditor ensuring that each r_i is generated correctly. To verify the consistency of a value at index $\mathbf{x}$ from epoch i to epoch j, the client simply queries the openings of $\overline{\text{rand}}_i(\mathbf{X})$ and $\overline{\text{rand}}_j(\mathbf{X})$ at $\mathbf{x}$. If $\overline{\text{rand}}_i(\mathbf{x}) = \overline{\text{rand}}_j(\mathbf{x})$, then with overwhelming probability, the value at index $\mathbf{x}$ has remained unchanged between epochs i and j. On the client side, the computational complexity of the consistency check amounts to two PCS verifications—independent of the number of epochs—which can be further reduced to a single opening if the underlying PCS supports homomorphism. On the server side, only two PCS openings at a Boolean point are required. Finally, the auditor's task is limited to verifying a single homomorphic relation and checking that the randomness r_i is correctly derived.

We highlight that all homomorphic relations can be check *non-homomorphically* in case the underlying PCS is non-homomorphic, via checking the polynomial relation at a random point. However, this requires the server to explicitly send extra PCS openings to the auditors, while it is for *free* in the homomorphic case.

4.5 Transparent dictionary construction

We generically present IRONDICT construction over a PCS in Figure 3. For simplicity, we assume the PCS supports homomorphic operations; if not, these operations can instead be carried out by having the auditor receive polynomial openings at a random evaluation point, as discussed in Section 4.4.

Theorem 1. *Let* PCS *be a secure multilinear polynomial commitment scheme. Then, the* IRONDICT *construction—outlined in Figure 3—achieves completeness and soundness in the random oracle model..*

We defer a formal proof of security to the full version of the paper.

Cost overview of IRONDICT. In the full version of the paper, we present a detailed cost table for each party in IRONDICT when instantiated with KZH-k. Due to space constraints, we omit this table here.

4.6 zk-IRONDICT

The goal of a *privacy-preserving* transparent dictionary [42, 11] is to ensure that commitments to the dictionary reveal no information about its underlying contents and that proofs provided by the server disclose only what is strictly necessary. This aligns with the notion of metadata privacy, as distinguished in prior work [13], which identifies two categories of privacy: (1) content privacy and (2) metadata privacy. Content privacy refers to access control policies enforced by the server—for instance, allowing users to retrieve only the public keys of their contacts, thereby mitigating spam (when the transparent dictionary is used as a PKI). These policies are external to the design of the transparent dictionary. In contrast, metadata privacy, which is the focus of a privacy-preserving transparent dictionary, aims to prevent leakage of information by the server, for example, as a result of publishing commitments on the bulletin board. Privacy is essential in settings such as secure messaging. If the server leaked patterns of value updates, a malicious party could infer which users frequently rotate their keys and which do not, potentially identifying less cautious users or rarely used devices. In another case, if an attacker compromises a device's value and

- $\mathsf{Setup}(\lambda,\mu) \to (\mathsf{key}_{\mathsf{Server}}; \mathsf{key}_{\mathsf{Auditor}}, \mathsf{key}_{\mathsf{Client}})$: Given a security parameter λ and the log of maximum supported size μ, it runs $\mathsf{Setup}_{\mathsf{PCS}}(\lambda,\mu)$ and output (vk, pk). Algorithm sets $\mathsf{key}_{\mathsf{Server}} := pk$, $\mathsf{key}_{\mathsf{Auditor}} := vk$ and $\mathsf{key}_{\mathsf{Client}} := vk$ and outputs these.
- $\mathsf{Server.Init}() \to (\mathsf{state}_0, \mathsf{com}_0)$: Given the dictionary size $N := 2^\mu$ and the PCS prover key, the server initializes three zero multilinear polynomials $\overline{\mathrm{index}}_0$, $\overline{\mathrm{value}}_0$ and $\overline{\mathrm{rand}}_0$, computes commitments to these polynomials and sets the tuple as com_0 and publishes com_0 on BB. At each epoch, state_i consists of the latest polynomials $\overline{\mathrm{index}}_i$, $\overline{\mathrm{value}}_i$, and $\overline{\mathrm{rand}}_i$. The commitment state, com_i, includes the PCS commitments to these polynomials.
- $\mathsf{Server.Update}(\mathsf{state}_i, \Delta) \to (\mathsf{state}_{i+1}, \mathsf{com}_{i+1}, \pi_{i+1})$: Parse Δ as $(\Delta_{\mathrm{index}}, \Delta_{\mathrm{value}})$ and update each polynomial as follows:
  - *Index dictionary:* Parse $\Delta_{\mathrm{index}} = \{\ell_j\}_{j\in[n_1]}$ and ensure all labels ℓ_j are not previously assigned indexes. Assign index $\mathbf{x}_j$ to each ℓ_j according to Figure 1 and define:

$$\overline{\mathrm{index}}_{i+1}(\mathbf{x}) = \begin{cases} H(\ell_j) & : \mathbf{x} = \mathbf{x}_j \\ \overline{\mathrm{index}}_i(\mathbf{x}) & : \text{otherwise} \end{cases}$$

    Compute commitment to $\overline{\mathrm{index}}_{i+1}$ (possibly homomorphically) and run a zerocheck for $\overline{\mathrm{index}}_i(\mathbf{x}) \cdot \left(\overline{\mathrm{index}}_{i+1}(\mathbf{x}) - \overline{\mathrm{index}}_i(\mathbf{x})\right)$, and output π_{register} which is essentially the zerocheck proof.
  - *Value dictionary:* Parse $\Delta_{\mathrm{value}} = \{(\ell_j, \upsilon_j)\}_{j\in[n_2]}$, retrieve indices $\mathbf{x}_j$ and define: $\Delta_i(\mathbf{x}) = \begin{cases} \upsilon_j & : \mathbf{x} = \mathbf{x}_j \\ 0 & : \text{otherwise} \end{cases}$

    Update $\overline{\mathrm{value}}_{i+1} = \overline{\mathrm{value}}_i + \overline{\Delta}_i$ and compute its commitment (possibly homomorphically), derive r_i and set $\overline{\mathrm{rand}}_{i+1} = \overline{\mathrm{rand}}_i + r_i \cdot \overline{\Delta}_i$ and again compute polynomial commitment to $\overline{\mathrm{rand}}_{i+1}$ too.

  It outputs new state state_{i+1}, commitment com_{i+1} and proof of epoch $\pi_{i+1} = (\pi_{\mathrm{register}}, \pi_{\mathrm{value}})$ where π_{value} is empty is the PCS is homomorphic, otherwise it contains PCS openings at a random point, used to evaluate the homomorphic relation, non-homomorphically.
- $\mathsf{Server.Lookup}(\ell, \mathsf{state}_i) \to (\mathsf{value}, \pi_{\mathsf{Lookup}})$: Retrieve index $\mathbf{x}$ corresponding to ℓ along with proof π_{index} that proves $\mathbf{x}$ is the valid index for label ℓ according to Figure 2. Generate PCS opening π_{PCS} that opens $\mathsf{value} = \overline{\mathrm{value}}_i(\mathbf{x})$ and finally output $(\mathbf{x}, \pi_{\mathrm{index}}, \pi_{\mathsf{PCS}})$.
- $\mathsf{Client.VerifyLookup}(\ell, \mathsf{com}_i, \mathsf{value}, \pi_{\mathsf{Lookup}}) \to 0/1$: Verify index proof using π_{index} as described in Figure 2 and value using PCS opening π_{PCS}, which essentially verifies that $\mathsf{value} = \overline{\mathrm{value}}_i(\mathbf{x})$.
- $\mathsf{Server.Consistency}(\ell, \mathsf{state}_i, \mathsf{state}_j) \to \pi_{\mathrm{consis}}$: This is similar to a lookup operation, but instead of opening $\overline{\mathrm{value}}_i$, we open $\overline{\mathrm{rand}}_i$ and $\overline{\mathrm{rand}}_j$ at the index. More precisely, retrieve index $\mathbf{x}$ corresponding to ℓ along with proof π_{index} that proves $\mathbf{x}$ is the valid index for label ℓ according to Figure 2. Generate PCS opening $\pi_{\mathsf{PCS},i}$ and $\pi_{\mathsf{PCS},j}$ that opens $\overline{\mathrm{rand}}_i$ and $\overline{\mathrm{rand}}_j$ at index $\mathbf{x}$ and finally output $(\mathbf{x}, \pi_{\mathrm{index}}, \pi_{\mathsf{PCS},i}, \pi_{\mathsf{PCS},j})$.
- $\mathsf{Client.VerConsistency}(\ell, \mathsf{com}_i, \mathsf{com}_j, \pi_{\mathrm{consis}}) \to 0/1$: Verify index proof using π_{index} as described in Figure 2 and value using PCS openings $\pi_{\mathsf{PCS},i}$ and $\pi_{\mathsf{PCS},j}$. Finally check that openings values are equal which essentially proves $\overline{\mathrm{rand}}_i(\mathbf{x}) = \overline{\mathrm{rand}}_j(\mathbf{x})$.
- $\mathsf{Auditor.VerifyInvariance}(\mathsf{com}_i, \mathsf{com}_{i+1}, \pi_{i+1}) \to 0/1$: Parse π_{i+1} as $(\pi_{\mathrm{register}}, \pi_{\mathrm{value}})$, now the auditor checks the two following checks: (1) verify π_{register} which is essentially verifying the zerocheck, and (2) recompute r_i through Fiat-Shamir heuristic and check that the following relations holds between polynomials underlying commitments $\mathrm{com}(\overline{\mathrm{value}}_i)$, $\mathrm{com}(\overline{\mathrm{value}}_{i+1})$, $\mathrm{com}(\overline{\mathrm{rand}}_i)$ and $\mathrm{com}(\overline{\mathrm{rand}}_i)$:

$$\overline{\mathrm{rand}}_{i+1}(\mathbf{X}) = \overline{\mathrm{rand}}_i(\mathbf{X}) + r_i \cdot \left(\overline{\mathrm{value}}_{i+1}(\mathbf{X}) - \overline{\mathrm{value}}_i(\mathbf{X})\right)$$

  This identity can be checked, for example, via homomorphism. Otherwise, it can be checked non-homomorphically by π_{value}, which includes opening of all four polynomials at a random point, as explained in Section 4.4.

Figure 3: IRONDICT construction based on a generic PCS.

observes that it remains unchanged in the next epoch, they can conclude that the device owner is unaware of the breach.

Privacy in previous work. CONIKs [42] described a notation of privacy which was later formalized by SEEMless [11] that defined privacy through a formal leakage function, ensuring that the information published by the server, namely, commitments posted on the bulletin board and accompanying proofs, reveals no more than what the leakage function allows. For instance, in SEEMless, the leakage function for value queries unnecessarily reveals the value's version number and the epoch of its last update. A further limitation of defining privacy through a leakage function is its fragility: the leakage must be carefully specified, and even minor changes to the system can alter it, requiring new proofs.

zk-IRONDICT's notion of privacy. IRONDICT is generically built on a PCS. The server reveals three main components: PCS commitments, PCS openings, and sumcheck proofs. Suppose the underlying PCS provides hiding commitments and zero-knowledge openings [9], and the sumcheck protocol is also zero-knowledge. In that case, each piece of published information proves only what is intended–nothing more. There are general compilers [15, 5] that can make sumcheck protocols zero-knowledge with minimal overhead. Our construction is concretely based on KZH, which by default does not support hiding. One could apply generic compilers designed to make homomorphic commitments hiding [9], but they involve generating and committing to randomness linear in the size of the polynomial. In the full version of the paper, we present a *zero-knowledge variant of KZH*, which may be of independent interest. This construction requires only $k^2 \cdot N^{1/k} + k \cdot N^{1/k}$ random scalars per opening on the server side. Specifically, $k^2 \cdot N^{1/k}$ scalars are needed to construct random polynomial r with $k \cdot N^{1/k}$ non-zero values together with its auxiliary inputs. Since there are $k-1$ auxiliary inputs, and each requires $k \cdot N^{1/k}$ scalar multiplications, the total cost amounts to $k^2 \cdot N^{1/k}$ for generating the polynomial commitment along with its auxiliary inputs. Importantly, this cost is independent of the opening point and can therefore be precomputed by the server. The remaining term, $k \cdot N^{1/k}$, corresponds to taking a random linear combination of the random polynomial with another polynomial and computing its opening at the client's desired point; this cost arises in the online phase. When applying zk-KZH, homomorphic relations over blinded commitments cannot be verified directly as in the unblinded case. Instead, they require an additional sigma protocol, which introduces only a small overhead for both the server and auditors wishing to verify such relations. Due to lack of space, we defer all the details to the full version of the paper.

Size leakage. Our privacy definition ensures that all public information posted by the server—such as commitments, sumcheck proofs, etc.—is zero-knowledge and reveals nothing beyond what it is intended to convey. Similarly, client queries (e.g., lookups and consistency proofs) reveal only the requested value in the dictionary. However, our construction leaks information about the *size* of the dictionary. In particular, the SRS inherently reveals an *upper bound* on the number of users, and the frequency of collisions in open-addressing leaks information about the current number of users. An adversary seeking information on the number of users could conduct probing attacks, registering multiple labels and, based on their identifiers (e.g., the number of hashes required to find an empty slot), infer statistical information on the current number of users. We leave analysis of such attacks to future work, noting that they do not contradict our zero-knowledge-based privacy definition. Furthermore, similar size information is leaked by Merkle-tree-based implementations. We consider it out of scope of our privacy definition.

5 Implementation and Evaluation

We implement and benchmark zk-IRONDICT in 4000 lines of Rust code[5]. Our implementation is built atop the `arkworks` [17] framework, and also uses the Hyperplonk [12][6] repository for the multivariate zerocheck and sumcheck implementations. For our PCS instantiation, we implement KZH-k [28] from scratch, with a tunable parameter k, in additional 2500 lines of Rust code, inspired by the original KZH implementation[7]. In our experiments, we set $k = (\log N)/2$ where N is the polynomial size. All server measurements are performed on a Google `c4-highmem-144-lssd` instance with 144 CPU cores and 1116 GB of memory. Client and auditor measurements are performed single-threaded on a MacBook Pro with 18 GB of RAM and a 12-core Apple M3 Pro CPU. We use the curve Bn254 for all our measurements, which is a standard pairing-friendly curve used in industry [10]. We employ a Schnorr signature-based VRF [51] in place of a hash function mapping arbitrary strings to points on the Boolean hypercube. All measurements are done over two dictionary sizes: (a) a small dictionary with 2^{24} (over 10 million) entries[8] for light-weight use cases such as software distribution (b) a large dictionary with 2^{30} (over 1 billion) entries for large-scale use cases such as messaging applications. Also, we report the numbers with an overprovisioning factor of $\alpha = 4$; meaning that the reported capacity utilizes $\frac{1}{4}$ of the implemented data structure, e.g. the polynomial underlying the dictionary of size 2^{30} is of size 2^{32}.

Remark 1. All reported numbers in this section, for both small and large dictionaries, correspond to zk-IRONDICT. While privacy is unnecessary for applications such as software distribution, we present the privacy-preserving results to

[5] https://github.com/alireza-shirzad/Iron-key
[6] https://github.com/EspressoSystems/hyperplonk
[7] https://github.com/h-hafezi/kzh_fold
[8] The number of apps on Google Play, Apple App Store, and the number of Linux packages are all below 5 million [47, 1].

illustrate how performance scales from 10 million to 1 billion entries. The main impact of adding privacy to IRONDICT is on server lookup and consistency proofs—for example, using IRONDICT instead of zk-IRONDICT with the small dictionary reduces server time for lookups and consistency proofs from 10ms to under 1ms, yielding a 10× improvement.

5.1 Server costs

We benchmark four important server operations: Index assignment, value update, lookups and consistency proofs. Also, we measure the size of the server key, which persistently resides in memory.

Index assignment time. We evaluate the time and communication cost (message size) incurred by the server when posting to the bulletin board to register a batch of new indices. We are primarily interested in the operational range under 1000 updates per second[9], and we mark this threshold in Figure 4. Within this range, the index assignment time is dominated by the time required to run a sumcheck prover on a polynomial of the dictionary size, which scales linearly with the dictionary size. After a certain point—outside the operational region—the cost of PCS opening begins to affect the overall registration cost which we can ignore. Figure 4 shows that for the small dictionary, the index assignment time is at around 10 seconds, while for the large dictionary, it is around 15 minutes.

Value dictionary update. Unlike the index dictionary, where update costs scale with the full dictionary size, updates to the value dictionary depend only on the batch size. The update time mainly reflects the cost of committing to a highly sparse update polynomial, which requires an MSM proportional to the batch size. As shown in Figure 4, for batches of hundreds to thousands, updates take around 2 seconds for the small dictionary and around 2 minutes for the large dictionary. The corresponding bulletin board message remains a small constant as well, under 1 kB per batch.

Lookups and consistency. Both lookup and consistency queries involve opening two polynomials at a point on the Boolean hypercube: $\overline{\text{value}}$ and $\overline{\text{index}}$ for lookups, and $\overline{\text{rand}}_i$ and $\overline{\text{rand}}_j$ for consistency. As a result, their server costs are nearly the same. Section 5.1 shows that for the small dictionary, the server time is around 15 milliseconds, while for the large dictionary it doubles to 30 milliseconds.

5.2 Client costs

We evaluate the cost of two types of client-side queries, namely: *lookup* and *value consistency*. As mentioned in Section 5.1, these operations have similar costs. In both cases,

[9]This threshold is higher than the peak registration rate observed in practice, such as Telegram's 70 million new users per day in 2021 (approximately 800 updates per second) [48].

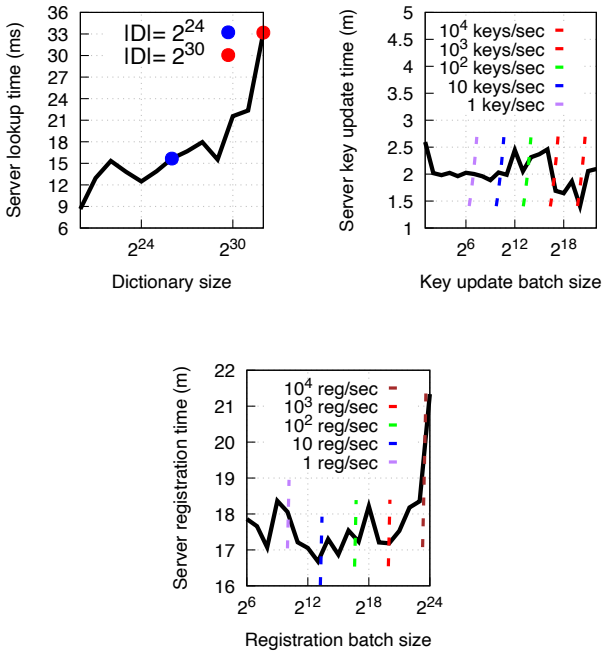

Figure 4: Server performance

the cost is dominated by two PCS verifications. This results in nearly identical performance for both query types. Figure 5 illustrates that for both small and large dictionaries, the lookup verification time is under 30ms. Also, the client key size is less than 1 MB. The communication cost for the lookup query, as illustrated in Figure 5 remains under 5 kB.

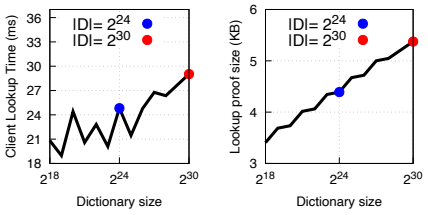

Figure 5: Client performance

5.3 Auditor costs

The auditor cost is dominated by the time taken to verify a PCS opening at a random point. There is also a logarithmic field operation cost for verifying the sumcheck proof, which is negligible compared to the PCS opening time. Figure 6 shows that for small and large dictionaries, the auditor can verify a single epoch less than 35 ms. This allows light-weight auditors, without needing to outsource the verification to a third party. Also Figure 6 shows that the audit proof size is less than 8 kB.

Protocol	Lookup time		Consistency check time		Auditor		
	Client (ms)	Server (ms)	Client (ms)	Server (ms)	Server time (s)	Client time (s)	Proof Size (MB)
WhatsApp AKD	0.5	1	8	13	60	11	1200
zk-IRONDICT$_k$	29	33	29	33	900	0.035	0.008

Table 1: Comparison of WhatsApp AKD over a dictionary of size 100 million and zk-IRONDICT over a dictionary of size more than 1 billion (2^{30}). For the consistency check of WhatsApp AKD, we assume that the key has been changed 10 times, i.e. its consistency check scales linearly with this number, while the consistency check of IRONDICT is independent.

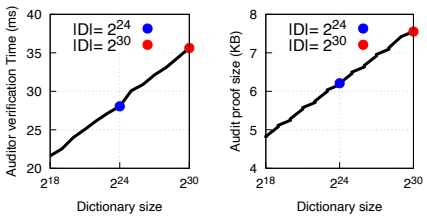

Figure 6: Auditor performance

5.4 Setup size

A crucial consideration in our scheme is the size and nature of the structured reference string (SRS). The small dictionary requires an srs of size less than 6 Gigabytes, while the large dictionary requires an SRS of size less than 350 Gigabytes. In practice, such SRSs are typically generated via multi-party ceremonies [66] that require at least one party to remain honest. The largest SRS generated that we are aware of has been done by Filecoin with a size of approximately 12GB [46]. The ceremony is designed to minimize communication among participants. Specifically, each participant obtains the current SRS state from the previous one, verifies its correctness via pairing checks, updates it, and then forwards it to the next participant. On Google c4-highmem-144-lssd with a cost rate of 11.36 USD per hour, generating the SRS for our large dictionary with 2^{30} users (i.e. the underlying polynomial of size 2^{32}) takes almost 1 hour, highlighting the practicality of the approach. For example, if 10 servers jointly perform the generation, the total cost would be about 110 USD and it takes around 10 hours because of the sequential nature of the ceremony. Finally, we emphasize that the KZH SRS is *updateable*, allowing it to be periodically refreshed through new ceremonies.

5.5 Comparison with WhatsApp AKD

To provide a concrete comparison with a state-of-the-art real-world system, we benchmark WhatsApp's authenticated key directory (AKD)—built on SEEMLESS [11] and Parakeet [40]—using their `akd` crate[10] on the same hardware. Despite using a machine with over a terabyte of memory, we were unable to scale the AKD implementation to one billion entries due to memory exhaustion; we therefore restricted our benchmarks to 100 million entries, which already places the evaluation in favor of WhatsApp.[11] We set throughput for both WhatsApp AKD and IRONDICT to 1000 new users and 1000 udpates per second.

Our results in table 1 demonstrate that zk-IRONDICT has 150000× smaller audit proof size and 300× faster audit verification time compared to WhatsApp AKD, effectively transforming WhatsApp's heavy server-driven auditing into a lightweight, self-auditable process for clients. This considerable improvement does increase the cost of lookups and consistency checks for both client and server, but all remain on the order of 30 ms, which constitutes an acceptable performance trade-off.

Remark 2. WhatsApp, being based on SEEMLESS and Parakeet, client ensures value consistency by retrieving the entire value history. In contrast, we compare our proof of consistency against WhatsApp's value-history approach. Unlike WhatsApp, IRONDICT's value history requires only a linear number of lookups and consistency proofs with respect to the number of value changes.

Acknowledgments

Hossein Hafezi conducted this work while hosted by Dario Fiore at IMDEA Software. Joseph Bonneau was supported by a16z crypto, NSF grant CNS-2239975, and gifts from Google, Meta, and Handshake Labs. The authors are grateful to Michael Rosenberg for his insightful feedback and valuable suggestions. We also thank Nirvan Tyagi for clarifying a detail related to VeRSA, and Pratyush Mishra for his assistance with questions regarding the implementation.

[10] https://github.com/facebook/akd

[11] In real-world deployments, the underlying Merkle tree is typically stored on disk, as it does not fit entirely in memory, and is accessed partially from disk, introducing extra costs. For simplicity, we omit these complications in our benchmarks.

Ethical Considerations

As with most cryptographic primitives, the technology is general-purpose. These are intended to improve accountability, integrity, and user privacy in large-scale infrastructures. We do not see direct ethical risks arising from this work.

Open Science

In accordance with the USENIX open science policy, we make our artefacts publicly available, and our codebase is accessible at the following stable and anonymous URL: https://github.com/alireza-shirzad/iron-key.

References

[1] 42matters. *Google Play vs iOS App Store: App stats and trends*. Accessed: 2025-08-30. Aug. 2025. URL: https://42matters.com/stats.

[2] Mustafa Al-Bassam and Sarah Meiklejohn. "Contour: A Practical System for Binary Transparency". In: *CoRR* abs/1712.08427 (2017). arXiv: 1712.08427. URL: http://arxiv.org/abs/1712.08427.

[3] Mashael AlSabah, Alin Tomescu, Ilia Lebedev, Dimitrios Serpanos, and Srini Devadas. "PriviPK: Certificate-less and secure email communication". In: *Computers & Security* 70 (2017), pp. 1–15. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2017.04.008. URL: https://www.sciencedirect.com/science/article/pii/S0167404817300834.

[4] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. "ARPKI: Attack Resilient Public-Key Infrastructure". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Association for Computing Machinery, 2014, 382–393. ISBN: 9781450329576. DOI: 10.1145/2660267.2660298. URL: https://doi.org/10.1145/2660267.2660298.

[5] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. "Aurora: Transparent Succinct Arguments for R1CS". In: *Advances in Cryptology – EUROCRYPT 2019*. Ed. by Yuval Ishai and Vincent Rijmen. Springer International Publishing, 2019, pp. 103–128. ISBN: 978-3-030-17653-2.

[6] Dan Boneh, Benedikt Bünz, and Ben Fisch. "Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains". In: *Advances in Cryptology – CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I*. Springer-Verlag, 2019, 561–586. ISBN: 978-3-030-26947-0. DOI: 10.1007/978-3-030-26948-7_20. URL: https://doi.org/10.1007/978-3-030-26948-7_20.

[7] Joseph Bonneau. "EthIKS: Using Ethereum to Audit a CONIKS Key Transparency Log". In: vol. 9604. Feb. 2016, pp. 95–105. ISBN: 978-3-662-53356-7. DOI: 10.1007/978-3-662-53357-4_7.

[8] Sean Bowe, Ariel Gabizon, Matthew Green, et al. *mpc: zk-SNARK parameter multi-party computation protocol*. https://github.com/zcash/mpc. Accessed: 2025-07-18. Oct. 2016.

[9] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. "Proofs for Inner Pairing Products and its Applications". In: *Advances in Cryptology – ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III*. Springer-Verlag, 2021, 65–97. ISBN: 978-3-030-92077-7. DOI: 10.1007/978-3-030-92078-4_3. URL: https://doi.org/10.1007/978-3-030-92078-4_3.

[10] Vitalik Buterin. "Ethereum: A next-generation smart contract and decentralized application platform". In: White Paper. 2014.

[11] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. "SEEMless: Secure End-to-End Encrypted Messaging with less Trust". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. Association for Computing Machinery, 2019, 1639–1656. ISBN: 9781450367479. DOI: 10.1145/3319535.3363202. URL: https://doi.org/10.1145/3319535.3363202.

[12] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. "HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. Lecture Notes in Computer Science. Springer, 2023, pp. 499–530. DOI: 10.1007/978-3-031-30617-4_17. URL: https://doi.org/10.1007/978-3-031-30617-4_17.

[13] Brian Chen, Yevgeniy Dodis, Esha Ghosh, Eli Goldin, Balachandar Kesavan, Antonio Marcedone, and Merry Ember Mou. "Rotatable Zero Knowledge Sets: Post Compromise Secure Auditable Dictionaries with Application to Key Transparency". In: *Advances in Cryptology – ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part III*. Springer-Verlag, 2022, 547–580. ISBN: 978-3-031-22968-8. DOI: 10.1007/978-3-031-22969-5_19. URL: https://doi.org/10.1007/978-3-031-22969-5_19.

[14] Weikeng Chen, Alessandro Chiesa, Emma Dauterman, and Nicholas P. Ward. *Reducing Participation Costs via Incremental Verification for Ledger Systems*. Cryptology ePrint Archive, Paper 2020/1522. 2020. URL: https://eprint.iacr.org/2020/1522.

[15] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. *A Zero Knowledge Sumcheck and its Applications*. Cryptology ePrint Archive, Paper 2017/305. 2017. URL: https://eprint.iacr.org/2017/305.

[16] Cloudflare. *Key Transparency Dashboard*. https://dash.key-transparency.cloudflare.com. Accessed: 2025-08-01. 2025.

[17] arkworks contributors. *arkworks zkSNARK ecosystem*. 2022. URL: https://arkworks.rs.

[18] Rasmus Dahlberg, Tobias Pulls, and Roel Peeters. *Efficient Sparse Merkle Trees: Caching Strategies and Secure (Non-)Membership Proofs*. Cryptology ePrint Archive, Paper 2016/683. 2016. URL: https://eprint.iacr.org/2016/683.

[19] Datareportal. *Digital 2024 Global Statshot (October): Worldwide Digital Overview*. Accessed: 2025-XX-XX. 2024. URL: https://datareportal.com/reports/digital-2024-october-global-statshot.

[20] Sascha Fahl, Sergej Dechand, Henning Perl, Felix Fischer, Jaromir Smrcek, and Matthew Smith. "Hey, NSA: Stay Away from my Market! Future Proofing App Markets against Powerful Attackers". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. 2014, 1143–1155. URL: https://doi.org/10.1145/2660267.2660311.

[21] Amos Fiat and Adi Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology — CRYPTO' 86*. 1987, pp. 186–194. ISBN: 978-3-540-47721-1.

[22] Dario Fiore, Dimitris Kolonelos, and Paola de Perthuis. "Cuckoo Commitments: Registration-Based Encryption and Key-Value Map Commitments for Large Spaces". In: *Advances in Cryptology – ASIACRYPT 2023: 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4–8, 2023, Proceedings, Part V*. Springer-Verlag, 2023, 166–200. ISBN: 978-981-99-8732-0. DOI: 10.1007/978-981-99-8733-7_6. URL: https://doi.org/10.1007/978-981-99-8733-7_6.

[23] Abba Garba Ph.D., Arne Bochem, and Benjamin Leiding. "BlockVoke -Fast, Blockchain-Based Certificate Revocation for PKIs and the Web of Trust". In: Nov. 2020. ISBN: 978-3-030-62973-1. DOI: 10.1007/978-3-030-62974-8_18.

[24] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. "Poseidon: A New Hash Function for Zero-Knowledge Proof Systems". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 519–535. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi.

[25] Hossein Hafezi, Alireza Shirzad, Benedikt Bünz, and Joseph Bonneau. *IronDict: Transparent Dictionaries from Polynomial Commitments*. Cryptology ePrint Archive, Paper 2025/1580. 2025. URL: https://eprint.iacr.org/2025/1580.

[26] Benjamin Hof and Georg Carle. "Software Distribution Transparency and Auditability". In: abs/1711.07278 (2017). arXiv: 1711.07278. URL: http://arxiv.org/abs/1711.07278.

[27] Yuncong Hu, Kian Hooshmand, Harika Kalidhindi, Seung Yang, and Raluca Ada Popa. "Merkle2 : A Low-Latency Transparency Log System". In: May 2021, pp. 285–303. DOI: 10.1109/SP40001.2021.00088.

[28] George Kadianakis, Arantxa Zapico, Hossein Hafezi, and Benedikt Bünz. *KZH-Fold: Accountable Voting from Sublinear Accumulation*. Cryptology ePrint Archive, Paper 2025/144. 2025. URL: https://eprint.iacr.org/2025/144.

[29] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*. Vol. 6477. Lecture Notes in Computer Science. 2010, pp. 177–194. DOI: 10.1007/978-3-642-17373-8_11.

[30] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil Gligor. "Accountable key infrastructure (AKI): a proposal for a public-key validation infrastructure". In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW '13. Association for Computing Machinery, 2013, 679–690. ISBN: 9781450320351. DOI: 10.1145/2488388.2488448. URL: https://doi.org/10.1145/2488388.2488448.

[31] Nikita Korzhitskii and Niklas Carlsson. "Revocation Statuses on the Internet". In: abs/2102.04288 (2021). arXiv: 2102.04288. URL: https://arxiv.org/abs/2102.04288.

[32] Nikita Korzhitskii, Matus Nemec, and Niklas Carlsson. *Postcertificates for Revocation Transparency*. 2022. arXiv: 2203.02280 [cs.CR]. URL: https://arxiv.org/abs/2203.02280.

[33] Ben Laurie. "Certificate transparency". In: (). URL: https://doi.org/10.1145/2659897.

[34] Sean Lawlor and Kevin Lewi. *Deploying Key Transparency at WhatsApp*. https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/. Engineering at Meta Blog. Apr. 2023. URL: https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/.

[35] Hemi Leibowitz, Haitham Ghalwash, Ewa Syta, and Amir Herzberg. *CTng: Secure Certificate and Revocation Transparency*. Cryptology ePrint Archive, Paper 2021/818. 2021. URL: https://eprint.iacr.org/2021/818.

[36] Julia Len, Melissa Chase, Esha Ghosh, Daniel Jost, Balachandar Kesavan, and Antonio Marcedone. "ELEKTRA: Efficient Lightweight multi-dEvice Key TRAnsparency". In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*. 2023, pp. 2915–2929. DOI: 10.1145/3576915.3623161. URL: https://doi.org/10.1145/3576915.3623161.

[37] Julia Len, Melissa Chase, Esha Ghosh, Kim Laine, and Radames Cruz Moreno. "OPTIKS: an optimized key transparency system". In: *Proceedings of the 33rd USENIX Conference on Security Symposium*. SEC '24. USENIX Association, 2024. ISBN: 978-1-939133-44-1.

[38] Derek Leung, Yossi Gilad, Sergey Gorbunov, Leonid Reyzin, and Nickolai Zeldovich. "Aardvark: An Asynchronous Authenticated Dictionary with Applications to Account-based Cryptocurrencies". In: *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Aug. 2022, pp. 4237–4254. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/leung.

[39] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. "Algebraic Methods for Interactive Proof Systems". In: *J. ACM* 39.4 (1992), pp. 859–868. DOI: 10.1145/146585.146605. URL: https://doi.org/10.1145/146585.146605.

[40] Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Oztürk, Kevin Lewi, and Sean Lawlor. "Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging". In: Jan. 2023. DOI: 10.14722/ndss.2023.24545.

[41] Sarah Meiklejohn, Pavel Kalinnikov, Cindy S. Lin, Martin Hutchinson, Gary Belvin, Mariana Raykova, and Al Cutter. *Think Global, Act Local: Gossip and Client Audits in Verifiable Data Structures*. 2020. arXiv: 2011.04551 [cs.CR].

[42] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. "CONIKS: Bringing Key Transparency to End Users". In: *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*. Ed. by Jaeyeon Jung and Thorsten Holz. USENIX Association, 2015, pp. 383–398. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/melara.

[43] S. Micali, M. Rabin, and S. Vadhan. "Verifiable random functions". In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*. 1999, pp. 120–130. DOI: 10.1109/SFFCS.1999.814584.

[44] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, Justin Cappos, and Bryan Ford. "CHAINIAC: proactive software-update transparency via collectively signed skipchains and verified builds". In: *Proceedings of the 26th USENIX Conference on Security Symposium*. SEC'17. USENIX Association, 2017, 1271–1287. ISBN: 9781931971409.

[45] Rasmus Pagh and Flemming Friche Rodler. "Cuckoo hashing". In: *J. Algorithms* 51.2 (May 2004), 122–144. ISSN: 0196-6774. DOI: 10.1016/j.jalgor.2003.12.002. URL: https://doi.org/10.1016/j.jalgor.2003.12.002.

[46] Protocol Labs. *Trusted Setup Complete*. https://filecoin.io/blog/posts/trusted-setup-complete/. Accessed: 2025-05-04. Aug. 2020. URL: https://filecoin.io/blog/posts/trusted-setup-complete/.

[47] Repology. *Raw repository package counts*. https://repology.org/repositories/packages. Accessed: 2025-08-30. n.d.

[48] Reuters. *Telegram Founder Says Over 70 Million New Users Joined During Facebook Outage*. Accessed: 2025-XX-XX. Oct. 2021. URL: https://www.reuters.com/technology/telegram-founder-says-over-70-mln-new-users-joined-during-facebook-outage-2021-10-05/.

[49] Michael Rosenberg, Tushar Mopuri, Hossein Hafezi, Ian Miers, and Pratyush Mishra. "Hekaton: Horizontally-Scalable zkSNARKs Via Proof Aggregation". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. CCS '24. Association for Computing Machinery, 2024, 929–940. ISBN: 9798400706363. DOI: 10.1145/3658644.3690282. URL: https://doi.org/10.1145/3658644.3690282.

[50] Mark Ryan. "Enhanced Certificate Transparency and End-to-End Encrypted Mail". In: Jan. 2014. ISBN: 1-891562-35-5. DOI: 10.14722/ndss.2014.23379.

[51] C. P. Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Ed. by Gilles Brassard. Springer New York, 1990, pp. 239–252. ISBN: 978-0-387-34805-6.

[52] Srinath Setty. "Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup". In: *Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III*. Springer-Verlag, 2020, 704–737. ISBN: 978-3-030-56876-4. DOI: 10.1007/978-3-030-56877-1_25. URL: https://doi.org/10.1007/978-3-030-56877-1_25.

[53] Trevor Smith, Luke Dickinson, and Kent Seamons. "Let's Revoke: Scalable Global Certificate Revocation". In: Jan. 2020. DOI: 10.14722/ndss.2020.24084.

[54] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. *Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning*. 2016. arXiv: 1503.08768 [cs.CR]. URL: https://arxiv.org/abs/1503.08768.

[55] Techi. *Pavel Durov said Telegram hits 1 billion users*. URL: https://www.techi.com/pavel-durov-said-telegram-hits-1-billion-users/.

[56] Thibault Meunier and Mari Galicer. *Cloudflare helps verify the security of end-to-end encrypted messages by auditing Key Transparency for WhatsApp*. Blog post on the Cloudflare Blog. Accessed 2025-07-31. Sept. 2024. URL: https://blog.cloudflare.com/key-transparency/.

[57] A. Tomescu. "PowMail: want to fork?: do some work". PhD thesis. Massachusetts Institute of Technology, 2015.

[58] Alin Tomescu, Vivek Bhupatiraju, Dimitrios Papadopoulos, Charalampos Papamanthou, Nikos Triandopoulos, and Srinivas Devadas. "Transparency Logs via Append-Only Authenticated Dictionaries". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. Association for Computing Machinery, 2019, 1299–1316. ISBN: 9781450367479. DOI: 10.1145/3319535.3345652. URL: https://doi.org/10.1145/3319535.3345652.

[59] Alin Tomescu and Srinivas Devadas. "Catena: Efficient Non-equivocation via Bitcoin". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 393–409. DOI: 10.1109/SP.2017.19.

[60] Alin Tomescu, Yu Xia, and Zachary Newman. *Authenticated Dictionaries with Cross-Incremental Proof (Dis)aggregation*. Cryptology ePrint Archive, Paper 2020/1239. 2020. URL: https://eprint.iacr.org/2020/1239.

[61] Nirvan Tyagi, Ben Fisch, Andrew Zitek, Joseph Bonneau, and Stefano Tessaro. "VeRSA: Verifiable Registries with Efficient Client Audits from RSA Authenticated Dictionaries". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. ACM, 2022, pp. 2793–2807. DOI: 10.1145/3548606.3560605. URL: https://doi.org/10.1145/3548606.3560605.

[62] Ioanna Tzialla, Abhiram Kothapalli, Bryan Parno, and Srinath Setty. "Transparency Dictionaries with Succinct Proofs of Correct Operation". In: *NDSS 2022*. July 2022. URL: https://www.microsoft.com/en-us/research/publication/transparency-dictionaries-with-succinct-proofs-of-correct-operation/.

[63] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. "SoK: secure messaging". In: *2015 IEEE Symposium on Security and Privacy*. 2015, pp. 232–249.

[64] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *Theory of Cryptography*. 2008, pp. 1–18. ISBN: 978-3-540-78524-8.

Protocol	Lookup		History		Auditor	
	Client	Server	Client	Server	Server	Upd. & Aud.
SEEMless [11]	$\log n^*$	$\log n^*$	$(u+\log T)\log n$	$(u+\log T)\log n$	$b\cdot\log(n+b)^*$	$b\cdot\log(n+b)^*$
Parakeet [40]	$\log n^*$	$\log n^*$	$(u+\log T)\log n$	$(u+\log T)\log n$	$b\cdot\log(n+b)^*$	$b\cdot\log(n+b)^*$
OPTIKS [37]	$u\log n^*$	$u\log n^*$	$u\log n$	$u\log n$	$b\cdot\log(n+b)^*$	$b\cdot\log(n+b)^*$
ELEKTRA [36]	$(u\log n+\log T)^*$	$(u\log n+\log T)^*$	$(u\log n+\log T)^*$	$(u\log n+\log T)^*$	$(b\cdot\log(n+b)+\log T)^*$	$(b\cdot\log(n+b)+\log T)^*$
Merkle^2 [27]	$\log n^2$	$\log n^2$	$\log n^2$	$\log n^2$	$\log n^{2*}$	$\log n^*$
VeRSA-IVC [61]	1	$(n/m+\log m)^*$	u	$(n/m+\log m)^*$	b	$\log T$
CONIKS [42]	$\log n^*$	$\log n^*$	naive	naive	$b\cdot\log(N+b)^*$	$b\cdot\log(N+b)^*$
IRONDICT$_k$	$k\cdot N^{1/k}$	$O(1)$	$u\cdot k\cdot N^{1/k}$	$u\cdot O(1)$	index assignment: N value update: b	$k\cdot N^{1/k}$
zk-IRONDICT$_k$	$k\cdot N^{1/k}$	$k^2\cdot N^{1/k}$	$u\cdot k\cdot N^{1/k}$	$u\cdot k^2\cdot N^{1/k}$	index assignment: N value update: b	$k\cdot N^{1/k}$

b: the number of elements added per epoch
n: the number of items in the dictionary
N: maximum dictionary's capacity
T: the number of epochs since the operation was last invoked
u: value's version
m: batch size (in case of VeRSA supporting batch lookups)
naive indicates that verifying history requires performing a lookup for each epoch. Complexities with star ($\star$) indicate the average case.

Figure 7: Asymptotic efficiency of transparent dictionary schemes: We denote our scheme as IRONDICT$_k$ when our construction is concretely initiated with KZH-k and zk-IRONDICT$_k$ our privacy-preserving variant. Since KZH-k uses precomputed arrays for PCS openings at boolean points, the server cost is $O(1)$. Update cost includes two terms: N (index assignment via cheap field operations) and b (value updates using group operations). The lookup and history cost for the server, which is $k^2\cdot N^{1/k}+k\cdot N^{1/k}$. The term $k^2\cdot N^{1/k}$ can be precomputed by the server since it is independent of the client query. We later expand on it in Section 4.6.

[65] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. "Doubly-Efficient zk-SNARKs Without Trusted Setup". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 926–943. DOI: 10.1109/SP.2018.00060.

[66] Faxing Wang, Shaanan Cohney, and Joseph Bonneau. *SoK: Trusted setups for powers-of-tau strings*. Cryptology ePrint Archive, Paper 2025/064. 2025. URL: https://eprint.iacr.org/2025/064.

[67] Wikipedia contributors. *Open addressing*. Accessed: 2025-08-18. 2025. URL: https://en.wikipedia.org/wiki/Open_addressing.

Appendix

A Tables