# SoK: Cryptographic Authenticated Dictionaries

Harjasleen Malvai, UIUC/IC3
Francesca Falzon, ETH Zürich
Andrew Zitek-Estrada, EPFL
Sarah Meiklejohn, University College London
Joseph Bonneau, NYU

***Abstract*—We systematize the research on authenticated dictionaries (ADs)—cryptographic data structures that enable applications such as key transparency, binary transparency, verifiable key-value stores, and integrity-preserving filesystems. First, we present a unified framework that captures the trust and threat assumptions behind five common deployment scenarios. Second, we distill and reconcile the diverse security definitions scattered across the literature, clarifying the guarantees they offer and when each is appropriate. Third, we develop a taxonomy of AD constructions and analyze their asymptotic costs, exposing a sharp dichotomy: every known scheme either incurs $\mathcal{O}(\log n)$ time for both lookups and updates, or achieves $\mathcal{O}(1)$ for one operation only by paying $\mathcal{O}(n)$ for the other. Surprisingly, this barrier persists even when stronger trust assumptions are introduced, undermining the intuition that "more trust buys efficiency". We conclude with application-driven research questions, including realistic auditing models and incentives for adoption in systems that today provide no verifiable integrity at all.**

## I. Introduction

What do encrypted-messaging PKI, filesystem integrity, blockchains, and secure cloud storage have in common? They all rely on the same cryptographic primitive: *authenticated dictionaries* (ADs). In its simplest form, an AD is a data structure that allows an untrusted server to commit to a set of key-value pairs (or equivalently, label-value pairs) using a short, constant-sized *commitment* (also sometimes called a *digest*). Given this commitment, the server can answer read queries for specific keys and convince an untrusting verifier of the correctness of a key-value binding, or handle write queries and provide a short proof that an updated commitment incorporates the requested change.

Different applications can use the same underlying construction by assigning different semantics to keys and values. A few such motivating applications include:

- **Database and filesystem integrity**, particularly when a large key-value database (with arbitrary semantics) is outsourced to an untrusted server [138], [156], [157], [153], or when ensuring the integrity of an entire filesystem [126].
- **Blockchains**, where ADs are used for tasks such as mapping accounts (keys) to balances or state (values), and committing smart-contract storage by mapping memory addresses (keys) to data (values), as pioneered by Ethereum [151].
- **Transparency systems**
  - **Key transparency**, where the AD maps user names (keys) to public keys (values), typically to support end-to-end encrypted communication [137], [113], [142], [20], [3], [28], [31], [98], [97]. Related proposals for secure messaging include key-usage transparency [155] and group-membership transparency.
  - **Binary transparency**, where the AD maps software application names (keys) to updates [52], packages [81], or builds [2], [121], typically in hashed form.
  - **Revocation transparency**, where the AD maps domain names (keys) to expired certificates (values), extending the earlier notion of certificate transparency (CT) to commit to all currently valid certificates for a domain [95], [119], [150], [58], [96], [88], [89], [118].

Although ADs were proposed nearly thirty years ago in the context of database integrity, most of their development and deployment over the past decade has occurred independently, driven by transparency systems and blockchains. Transparency systems were first proposed to detect misbehavior by certificate authorities (CAs), which vouch for mappings between web domains and public keys used to establish secure TLS sessions. An incorrectly issued (or "rogue") certificate from any single CA can undermine user privacy and security [49], [139], and there have been many prominent CA compromises [66], [55], most notably DigiNotar [57] and Comodo [134] in 2011. CT has now been successfully deployed in practice for over a decade, with modern browsers rejecting any certificate that has not been publicly logged. While CT itself does not implement a full AD and instead commits only to the set of valid certificates rather than a domain-to-certificate mapping (technically, CT is a vector commitment), it directly inspired later work on transparency in other domains. In parallel, many blockchains such as Ethereum utilize ADs to efficiently commit on-chain to user and smart-contract state.

Despite underpinning critical systems ranging from end-to-end encrypted messaging and filesystem integrity to blockchains and secure cloud storage, AD constructions remain fragmented across disparate trust models, workloads, and APIs. This fragmentation forces developers to reimplement core functionality, complicates rigorous evaluation, and impedes secure, large-scale deployment. In this work, rather than presenting new benchmarks, we deliver the first comprehensive framework for ADs: a unified terminology and five core trust models that clarify the entire design space. Our framework lays the conceptual foundation for standardized APIs, modular

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.241465
www.ndss-symposium.org

implementations that integrate seamlessly into diverse applications, and meaningful future artifact-based comparisons—paving the way for faster innovation, stronger security guarantees, and interoperable deployments across domains.

**Our contributions.** In this work, we provide:
**(C1) A unified framework for trust models.** We catalog five core roles (server, client, data source, auditor, public bulletin board) and show that every AD in the literature fits into one of five models differing in who is trusted and for what. For example, cloud-based logs such as Amazon QLDB [8] use a trusted-source model, whereas key-transparency systems like WhatsApp do not have a single authoritative source.
**(C2) A comparative map of security definitions.** We distill the often informal security notions in the AD literature into three core definitions which we call *value binding*, *history binding*, and *read-write consistency*, and analyze their logical relationships and prerequisites. This allows practitioners to select the definition that best enforces their real-world safety goals while balancing usability requirements. For example, read-write consistency only applies when writers manage cryptographic secrets; thus, in key-transparency settings, history binding is the strongest applicable notion.
**(C3) A taxonomy of construction styles.** We identify the approaches to instantiate ADs in the literature, provide a classification of works along this taxonomy, and discuss their various advantages and disadvantages.
**(C4) An asymptotic performance survey.** Our analysis of more than 30 schemes that innovate on AD design, introduce novel data structures, or add new functionality reveals a striking bifurcation: either both lookup and update run in $\mathcal{O}(\log n)$ time, or updates are $\mathcal{O}(1)$ while lookups degrade to $\mathcal{O}(n)$ (or vice versa). Crucially, stronger trust assumptions do not automatically yield better asymptotics: several no-trusted-party designs match the update/lookup trade-offs of trusted-data-source ADs. This challenges the common engineering intuition that centralized trust necessarily leads to better performance.

**Paper outline.** The remainder of this paper is organized as follows. In Section II, we define the paper's scope, describe our two-step paper-gathering methodology, and introduce the evaluation criteria underlying our comparison. We also summarize related work that we consider out of scope and explain our rationale. In Section III, we develop a unified model of ADs, first by introducing the core roles and operations (Section III-A) and then presenting five trust models (Section III-B). In Section IV, we present an API for ADs along with three security definitions–value binding, history binding, and read-write consistency–and discuss their implications. Section V summarizes the technical building blocks of ADs, categorizes constructions by their technical instantiation, surveys the literature within this framework, and presents our asymptotic performance analysis, highlighting the $(\log n, \log n)$ versus $(1, n)$ trade-off. Finally, in Section VI we discuss deployment lessons, open challenges for systems builders, and directions for future work.

## II. Related Work and Evaluation Criteria

In this section, we discuss work related to ADs that we deem out of scope. We also discuss our evaluation criteria and methodology for finding the constructions we do include.

### A. Scope, evaluation criteria and methodology

Here, we discuss our scoping, methodology, and the primary criteria we use to compare AD constructions (a detailed comparison is provided in Table II).

**Scope.** Our goal is to systematize the literature on ADs, which are cryptographic data structures that enable an untrusted server to prove the integrity of a key-value store via succinct commitments. We only include papers whose stated contributions lie in the design, analysis, or deployment of an AD construction, and we omit works focusing solely on related primitives (e.g., logging, pure accumulators, general vector commitments, gossip protocols, etc.) unless they present a novel approach to realizing or optimizing an AD.

**Paper gathering methodology.** We assembled our bibliography using a two-stage process. First, we manually reviewed well-known recent papers such as [82], [107] and their closely related works, as well as the Google Scholar pages of their authors. To ensure completeness, we also developed a Python script that leverages the Semantic Scholar API. The script scanned major security, database, and cryptography venues over the past two decades, filtering by keywords such as "authenticated dictionary" and "verifiable directory." Both the script and detailed filtering criteria are provided as a reusable artifact [106] (see App. B for details).

**Efficiency.** Efficiency is the primary practical consideration for most applications of interest, which often require scaling to millions or billions of entries. We evaluate efficiency along three dimensions: (1) server costs for database updates and lookups; (2) client costs for verification and proof size (the latter is asymptotically equivalent to verification time and thus omitted); and (3) costs for auditing and monitoring. We present asymptotic comparisons in Table II. While concrete overheads are important in practice, a fair empirical comparison is currently infeasible. Many older constructions (particularly from the 1990s and early 2000s) lack public or maintained implementations. More importantly, the space is fragmented across threat models, workloads, and APIs, making even conceptually similar systems difficult to compare experimentally.

Our paper identifies five distinct trust models, ranging from settings with a trusted data source to fully decentralized transparency systems. Some of these models are directly comparable, for example, the trusted-source models form a comparable cluster, and the no-trusted-source models another. We hope that the unified terminology and abstractions we introduce will help bridge these design spaces, enabling more modular constructions and clarifying tradeoffs across settings. In the long term, this could facilitate interoperable APIs and more meaningful benchmarking across applications. As of today, the only such benchmarking effort we are aware of is

VeriBench [157], which focuses on database integrity, a relatively mature use case with more standardized assumptions, but even VeriBench benchmarks only six implementations.

**Required cryptographic assumptions.** All AD constructions require a collision-resistant hash function (CRHF) and some require no further cryptographic assumptions. Others, which we deem algebraic constructions, require additional cryptographic assumptions e.g., the strong RSA assumption. We provide a detailed list in App. C. Practitioners generally prefer systems with fewer and better-studied assumptions as they are less vulnerable to improved cryptanalysis. Furthermore, such systems typically admit simpler, more well-studied implementations, which is important for robustness.

### *B. Other authenticated data structures*

**Proof of Liabilities/Reserves.** Proofs of Liabilities and Proofs of Reserves allow a prover to commit to its total liabilities or assets and efficiently demonstrate to each user that their balance is included [110], [47], [48], [124], together forming a Proof of Solvency [38], [135], [26]. These schemes often employ Merkle or Verkle trees and RSA accumulators, but require additional guarantees beyond standard ADs, such as proving the sum of individual values, and typically use zero-knowledge proofs to show inclusion of each user's assets or liabilities. Because their functionality and security goals differ from those of ADs, we exclude them from this study.

**Accumulators, VCs, & memory checkers.** Memory checkers allow a client to outsource an array and verify that reads reflect the most recent writes [16]. Vector commitments (VCs) commit to a vector and allow opening at specific indices [27], [123]. Authenticated hash tables [130] and set accumulators [120], [13], [43], [63] support verifiable membership or non-membership queries. These primitives provide strictly less functionality than ADs and can be composed into or derived from ADs (e.g., via the reductions of Falzon et al. [53]); therefore, they are considered out of scope for this study.

**PDP and POR.** Provable Data Possession (PDP) is a restricted form of memory checking that allows a client to outsource data to an untrusted server and verify data posession without retrieval [7]. Proofs of Retrievability (POR) were introduced concurrently and go further by enabling the server to prove not only possession but also that the client can retrieve the entire file [85]. While both PDP and POR are related to ADs, they offer more limited functionality and are out of scope.

**Data structures for complex operations.** Authenticated data structures have been proposed to support more complex queries, such as graph connectivity and geometric queries [71], shortest-path queries [154], subgraph similarity search [132], prefix authentication [115], and wildcard prefix lookups [39]. Their enhanced functionality and/or differing security goals place these works outside the scope of our study.

### *C. Logging and related solutions*

**Early solutions.** One of the earliest applications to consider authenticated mappings was document time-stamping, beginning with the work of Haber and Stornetta [78]. Their main goal was to commit to the state of a user's document at a specific point in time, which they achieved using a hash chain. Subsequent works sought to improve the efficiency of this construction [12], [11], [13], [25], [24]. See, for example, Lipmaa [104] for a survey. As Table II shows, the key advantage of more recent schemes is that they reduce the cost of lookups to (at least) sublinear in the number of entries.

**Chronological logging.** Chronological logging extends early research and supports only insertions of new keys, with no updates to existing keys. Conceptually, it resembles an AD in which the keys are indices of log entries, akin to Lamport timestamps [93]. For example, a log entry might consist of a statement, $\mathsf{stmt}$, such as "Cert $C$ issued for domain $D$," and if this is the $i$th statement, key $i$ maps to $\mathsf{stmt}$. To determine the current state associated with domain $D$, a party must traverse the entire mapping to collect all relevant statements $\mathsf{stmt}_1^D, \ldots, \mathsf{stmt}_k^D$ and derive $D$'s value. If domains themselves were treated as keys with mutating values, verifying the state of any particular key would be linear in $N$, the number of updates in the AD. While such solutions have not appeared explicitly in the AD literature, related work includes certificate transparency [94], [40], [137], [46], [83], [91], [44], [152], tamper-evident or transaction logging [140], storage timestamping [125], and package manager transparency [81], [2], [79]. Due to the limitation on the set of keys, these are more limited than ADs and we do not discuss them further. Chronological logs are sometimes combined with a full AD solution to support efficient updates or patch an existing system; in such cases, we *do* consider them.

**Certificate transparency (CT).** CT is an application that uses chronological logging to record all certificates issued by a certificate authority. It is one of the most widely deployed AD-related applications in cryptography for ensuring the integrity of TLS certificates. Thus, AD applications can learn from CT, including challenges related to sharing commitments [111] and privacy-preserving reporting and querying [41], [50], [87].

### *D. Consensus and related problems*

**Dissemination mechanisms or public bulletin boards.** Certain AD models assume the existence of a mechanism that allows users to obtain consistent views of small cryptographic commitments to the dictionary. We refer to this mechanism as a *public bulletin board* (PBB) or, more generally, a *dissemination mechanism* (DM). In practice, bulletin boards for ADs and closely related transparency systems have been instantiated using gossip protocols [36], [112], [29], [42], [103], [108], blockchains [145], [20], [2], other decentralized systems [23], or reliable broadcast and witnessing mechanisms [107], [141], [121]. To keep the scope of this work focused on ADs across different trust models, we leave a general security definition and a systematization of dissemination mechanisms and public bulletin boards for transparency applications to future work.

**Blockchain-related models.** A blockchain or decentralized system, especially one that supports Turing-complete smart

contracts [151], can be viewed as an AD. In this setting, the AD is replicated across many servers (or *full nodes*) that respond to lookup queries. However, the blockchain literature and related areas largely focus on a specific aspect of the update mechanism: no fixed party is trusted to perform writes—an assumption that necessitates Byzantine consensus. Layer-2 blockchains are often equivalent in construction to our transparency model, particularly when implemented in a centralized manner. At the same time, layer-2 solutions typically aspire to decentralization and rely on the main chain as a fallback mechanism to prevent or recover from errors; consequently, much of the layer-2 literature focuses on this interaction with the main chain. *Light clients* are restricted to lookup functionality and therefore do not support a full-fledged AD. A separate line of work on stateless blockchains (e.g., [143], [18], [35], [34]) primarily focuses on vector commitments and accumulators, with an emphasis on efficiently updating proofs for subsets of values. Overall, while blockchain and adjacent research heavily relies on ADs, these works typically treat ADs as a black box or employ alternative data structures rather than innovating on AD design itself. We therefore include blockchain-related works only when they introduce new AD constructions or insights, and otherwise refer the reader to existing surveys on light clients [30] and layer-2 solutions [77].

### E. Other related works

**Frameworks.** In this work, we restrict ourselves to key-value stores and closely related authenticated data structures. There exists, however, a broader literature on authenticated variants of more general data structures. For further discussion, we defer to prior work [67], [117], which presents generic compilers for transforming arbitrary data structures into authenticated ones. More recently, LegoLog [54] proposes a configurable framework for transparency systems with a focus on optimizing for different workloads. LegoLog is built around a specific construction, Merkle$^2$, which we discuss in Sec. V, but could likely be extended to support other AD constructions within a similar model. VeriBench [157] provides a framework for benchmarking verifiable database implementations; however, it does not analyze asymptotic performance or consider older constructions that lack modern implementations, but might perform well if implemented today.

**Related SoKs.** The SoK of Brandt et al. [22] surveys a subset of the literature included here, limiting themselves to key transparency and providing a new UC security definition for it. We refer the reader to the SoK of Hicks [80] for further discussion of specific applications, legal, and organizational aspects of transparency systems and their relation to traditional offline notions of transparency. Another SoK [111] focuses on the privacy issues that appear in practice when clients wish to obtain proofs and raise alarms in certificate transparency. These works raise important questions about specific applications that may employ ADs, but they do not focus on the AD primitive itself or its associated trust models.

## III. Modeling Authenticated Dictionaries

We survey five distinct AD trust models: two that assume a single trusted data source, one with multiple data sources each trusted for a subset of the dictionary, and two that eliminate any root of trust. We show how each model offers different guarantees and applies to different deployment scenarios. Fig. 1 shows these models and their relationships, highlighting which schemes interoperate without additional cryptography.

### A. Roles and Operations in ADs

We first discuss the most general set of roles across the AD literature and provide a unified terminology. Note that the exact trust assumptions may vary across the different models and some of the roles may be omitted.

**Server.** The *server* stores and updates a queryable database, typically modeled as a key-value store (though some systems use alternative models, e.g., a chronological list in CT). Since the server is untrusted, it uses an *authenticated data structure* to maintain integrity, and, in some applications, transparency under specified invariants (e.g., monotonic version numbers).

The authenticated data structure enables succinct *commitments* to the database state, which are periodically computed and signed. In some settings, like cloud storage, the data owner who serves as a *trusted data source* helps compute or verify commitments and signs them. In transparency applications with no trusted source, the server signs and disseminates commitments to clients—typically via a public bulletin board (see below). In such cases, the server must also prove that each commitment is syntactically correct and that updates since the previous commitment follow the prescribed protocol and satisfy required predicates. Additionally, the server should efficiently prove membership (or non-membership) of a key-value pair with respect to the latest commitment, and clients should be able to verify these proofs efficiently. To support tracking and consistency, the commitments themselves may be organized in an authenticated data structure, such as an authenticated linked list [113] or Merkle tree [112], to enable verifying both key-level queries and global state changes.

**Client.** A party, referred to as the user or *client*, can query the database hosted by the server and request updates to its values. In some settings, there is a single client; in others, multiple clients may interact with the database. In multi-client scenarios, updates are often subject to rules around authentication and access control (e.g., a user can only update their own values) and in some cases there may also be restrictions on which keys a user is permitted to query. The server is responsible for enforcing these rules. When querying the database, clients should be able to efficiently verify whether an entry $(k, v)$ is a (non-)member of the database with respect to the latest commitment.

**Data source.** In many non-transparency applications of ADs, it is typically assumed that a *trusted data source* exists i.e., a party from which the ground truth database and all updates originate. This source may be the client itself or a separate trusted entity. For instance, in a privately outsourced database,

the client acts as both the data source and the querying party. In a multi-client setting, the data source might be an external entity. When such a trusted data source exists, the data source typically makes updates to the database, computes the commitments, and signs the commitments.

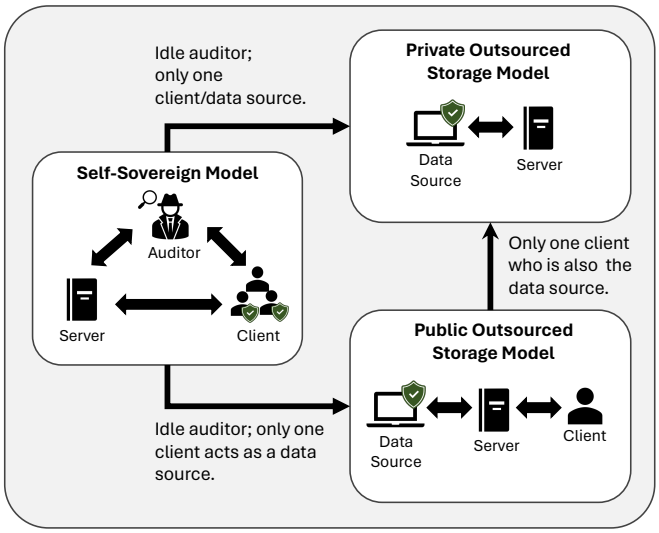
**Auditors.** In some applications, such as transparency systems, additional parties called *auditors* verify that the server maintains the required invariants, which vary depending on the construction. Auditors were initially assumed to perform relatively heavy computations (e.g., linear in the dictionary size). They perform purely syntactic checks and do not need to understand the specific context of the application the AD supports. By checking these invariants, auditors can generate cryptographic evidence of server misbehavior, ensuring detection as long as at least one auditor remains honest.

Some recent constructions (e.g., [82], [147], [148]) aim to enable efficient client-side auditing, a property called *client auditability*. Certain algebraic approaches achieve this naturally using compact, append-only proofs [147], [144], while others rely on SNARK-based verifiable computation performed by the server [61], [74]. A stronger alternative enforces correct updates through smart-contract-enabled blockchains [20].

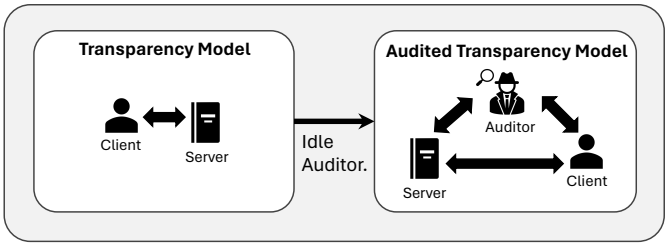
**Monitors.** In certain applications, *monitors* inspect dictionaries for unexpected or incorrect values and perform domain-specific semantic checks that require application-level understanding. Unlike auditors, monitors typically cannot produce cryptographic proofs of misbehavior; instead, they can only alert affected users or authorities.

Some constructions assume that dedicated monitors inspect every database entry, while others allow users to efficiently monitor only their own entries [28], [107], [97]. Hybrid approaches also exist. For example, Dahlberg et al. [40] present a lightweight system for *verifiable monitoring* of certificate transparency logs. Other constructions [112], [121], [97], [86] commit to the history of each key's values, reducing monitoring to simple lookups of the most recent epoch. While monitors can alert users or authorities to detected misbehavior, they generally cannot provide cryptographic proof to convince third parties (e.g., only a domain operator knows whether a certificate should have been issued for a given domain).

**Public bulletin board.** ADs without a trusted party require a *public bulletin board (PBB)* to ensure that all clients observe the same commitment to the AD's latest state. Without a PBB, a malicious server could *equivocate* by presenting different commitments (e.g., binding to different public keys) to different clients in a middle-person attack; the PBB's core function is therefore *anti-equivocation*. Importantly, clients cannot rely on the application's native communication channel (e.g., within a messaging system) to compare commitments, since a compromised server could block, delay, or tamper with these messages. Detection instead requires an *out-of-band* channel. For PBB instantiations, see Sec. II-D.




(a) Trusted Data Source




(b) No Trusted Party

Fig. 1. An overview of the authenticated dictionary models and their relationships. A green checkmark (🛡) next to a party denotes that it is trusted. An arrow from one category to another means that constructions that fit the first model can be adapted to work in the second model without additional cryptographic assumptions.

### B. Authenticated Dictionary Models

We classify ADs into five models–three with a trusted data source and two without–based on a survey of over 30 papers that include an innovation on an authenticated dictionary. For each model, we detail its threat assumptions, parties, applications, and list representative works and deployments below and summarize it in Table I. Fig. 1 illustrates these models, and Sec. VI-B explains how to compile between them.

**A note about trust.** We use the term "trust" in two ways:
(a) *Cryptographic trust*, where a party is assumed to correctly manage cryptographic keys and digitally sign its operations. Any operation signed by a "trusted" party is assumed correct. For example, in a model with a single trusted data source, this party signs all updates to the AD. Since all operations are signed, misbehavior can be publicly and reliably proven.
(b) *Non-cryptographic trust* includes assumptions that cannot be enforced or verified cryptographically. For example, in a "no trusted party" model, if the server enforces access control, there is no cryptographic way to prove that it performed this function correctly and did not, for instance, disclose the data associated with a key to an unauthorized user. With non-cryptographic trust, *false accusation* is another kind of "at-

tack". If the value associated with a label owned by a specific client is modified by the server, the client *can* detect this change. However, any claim that the change was unauthorized has no cryptographic proof. Thus, a client can claim a detected change was unauthorized and third parties have no way to ascertain the truth of this accusation. False accusations might be used to decrease public trust in a targeted server.

*1) Private Outsourced Storage Model:* This model involves two parties: (1) a *data source* (which acts as the client[1]) defining the dictionary and (2) a server hosting it. Security requires that for any lookup of a label, if the lookup verifies, the response is the corresponding set of values defined by the data source. This model parallels memory checking [15], [16] but differs by supporting arbitrary key-value mappings.

**Cryptographic trust assumptions.** A single trusted data source, holding a constant-sized commitment in secure local storage, is the origin of all data and updates. Since this party is also the client in this model, it must securely manage its own cryptographic secrets and corresponding verification data. Since only the data source can query or modify the dictionary, no PKI is needed. A caveat is that if the data source's signing key is lost or compromised, updates become impossible without an external key-recovery mechanism.

**Non-cryptographic trust assumptions.** The untrusted server is trusted for availability. Any other misbehavior is detected by the data source/client cryptographically.

**Applications.** This model has relatively limited applications, most of which pertain to outsourcing personal or organizational data to cloud providers. Typical examples are encrypted file storage or cloud-hosted databases, especially when clients have limited local storage or require long-term archiving.

**Constructions.** Many pioneering AD works fall into this model [122], [70], [109], [129]. Variants such as authenticated hash tables [130] and streaming authenticated data structures [128] also fit within this model, and Miller et al. [117] provide a general compiler for converting arbitrary data structures into ADs under these assumptions.

*2) Public Outsourced Storage Model:* This model comprises of three parties: (1) a data source from which the data originates, (2) a server that hosts the database, and (3) a client that queries the data. As in the previous model, security requires that any label lookup whose proof verifies returns exactly the value defined by the data source.

**Cryptographic trust assumptions.** A trusted data source defines and signs the dictionary along with its periodic commitments, then replicates both to one or more untrusted servers while retaining some or all of the dictionary locally. Clients verify responses using the source's public key, requiring no additional state, so this model relies on a PKI without the need for the source to remain online for lookups. A notable caveat is that updates, and possibly verification, become impossible if the trusted data source's key is lost or compromised.

**Non-cryptographic trust assumptions.** In this model, servers are assumed to respond to client lookups with values, proofs, and the signed, timestamped commitment; servers are trusted only for availability.

**Applications.** Schemes in this model are ideal for remote storage and scenarios where third-party clients need to query outsourced data, such as ensuring the integrity of legal documents, medical records shared across hospitals, company credit card transactions subject to financial audits, or government records accessed by multiple organizations. Some early work on CT also adopted this model, where the certificate authority acts as the trusted data source (e.g., [122]).

**Constructions.** Many works that proposed Private Outsourced Storage model schemes also described schemes for this model, e.g. [130], [70], [122], [129]. Other examples that fall within this three-party setting include [4], [37], [68] as well as the schemes Balloon [133] and Insynd [131].

**Instantiations.** Amazon QLDB provides a centralized ledger with a Merkle tree–based immutable log owned by a trusted authority; clients fetch the root hash and verify the transaction history [8]. immudb offers an append-only Merkle tree log supporting both key–value and SQL data, with data owners signing each root so third-party clients can later verify the authenticity of returned roots or individual records [84], [127]. Both systems fit the Public Outsourced Storage model and can be adapted for the Private Outsourced Storage model by merging the data source and client roles (see Section VI-B).

*3) Self-Sovereign Model:* This model comprises three types of parties: (1) multiple clients, (2) a server, and (3) an auditor. Informally, security guarantees that any server misbehavior is detectable by honest clients or auditors.

**Cryptographic trust assumptions.** Each client owns and signs updates for its own key-value pairs, making it the trusted source for those keys. Clients must manage their private keys (losing a key affects only its associated label). Unlike transparency models (see Sec. III-B4), which detect unauthorized changes without cryptographic proof, this model ensures verifiable updates at the granularity of individual labels. Unlike the Public/Private Outsourced Storage model, where a single data source holds one key, here each client's key secures only its own label, so losing a key only disables updates for that label.

**Non-cryptographic trust.** An untrusted server hosts the dictionary, responds to client queries i.e., is trusted for availability. Note that this model generally assumes either an out-of-band mechanism (e.g. akin to a PKI) or trusts the server for correctly binding client public keys with their respective labels.

**Applications.** This model fits transparency settings where clients manage their own keys (e.g., news and binary transparency) and is less suitable for key-transparency scenarios.

**Constructions.** Examples include Merkle$^2$ [82], which uses nested Merkle trees for transparency logs, and Goodrich

[1]In earlier literature, the Private Outsourced Storage and Public Outsourced Storage models are sometimes simply referred to as the *two-party* and *three-party* models, respectively. The client is also commonly referred to as the *data consumer*; see [130] for reference.

et al.'s multi-source AD schemes [69]. CONIKS [113] and SEEMless [28] similarly introduce variants with "super-users" who manage update signing keys for their usernames.

**Instantiation.** The secure messaging and file-sharing service Keybase [86] implements this model via per-user signed *sigchains*–ordered lists of statements (e.g., key additions, follows)–anchored in a public Merkle tree to prevent rollback and ensure verifiable key transparency.

*4) Audited Transparency Model:* This model involves three parties: (1) one or more clients, (2) a server, and (3) one or more auditors. Security requires that if at least one honest auditor successfully verifies a commitment, consistency must hold between consecutively published commitments. Similarly, any lookup that verifies for an honest client must align with the commitment history, ensuring all honest clients share the same current view. Many schemes in this model impose additional security requirements, such as the *append-only* property, which ensures that existing records are immutable and never deleted.

**Cryptographic trust assumption.** The server is trusted to hold a signing key with which it signs all messages and commitments. Here we also include Aardvark [100], which uses three parties: a trusted validator, a client, and an untrusted archive i.e., server. The validator both publishes signed commitments (acting as a bulletin board) and audits updates, while clients query the archive using those commitments.

**Non-cryptographic trust assumptions.** There is no trusted data source and any party may arbitrarily deviate from the protocol.[2] The server is assumed to store the entire dictionary, whereas auditors and clients only store a short commitment (not necessarily the same one). This model covers schemes in which auditors play a fundamental role and do not simply implement a cryptographic building block like a bulletin board (e.g., [147]). Third-party auditors check the well-formedness of the data structure and updates, and whistle-blow if some update is malformed. The mechanism for obtaining the commitment is generally out-of-band, but this commitment is assumed to be signed by the server. An individual client's claim about a malicious update for its label is not cryptographically verifiable, but inconsistent views for the same label or for the commitment at the same timestep are.

**Applications.** This model underpins transparency systems with active auditors and can also reduce storage requirements for blockchain validators.

**Constructions.** Examples are CT [94], SEEMless [28], and Parakeet [107].

**Instantiations.** WhatsApp's key transparency based on Parakeet [107], [101] uses an append-only auditable key directory plus a third-party audit record that anyone (e.g., Cloudflare) can verify independently [114]. Proton AG's email service employs a Merkle-tree directory whose root is embedded in a CA-signed certificate with SCTs, letting clients verify directory integrity [1]. Other examples include Google's key transparency [72] and Apple's CT [5].

*5) Transparency Model:* This model involves only two parties: (1) a client and (2) a server. The security goal typically requires that any lookup query successfully verified by an honest client must be consistent with the view of other honest clients, or that any misbehavior will eventually be detected.

**Cryptographic trust assumptions.** The server is trusted to hold a signing key with which it signs all messages and commitments. All honest clients maintain a consistent view of the dictionary and can detect any deviation by the server.

**Non-cryptographic trust assumptions.** Neither server nor client is trusted; both may deviate arbitrarily. The server holds the full dictionary, while each client stores a short commitment fetched from a public bulletin board. Clients self-audit their own entries (and optionally others') to ensure consistency. Thus, if the server makes a malicious update but serves the same update to all users, it's not cryptographically verifiable. However, inconsistent claims made at the same time, or malformed commitments or updates are verifiable.

**Applications.** This setting is best tailored to transparency applications such as certificate transparency. Note, however, that the original certificate transparency scheme [94] does not fall into this category since it also includes third-party auditors, as explained in Sec. III-B4.

**Constructions.** Schemes that fall into this model include AAD [144] and VeRSA [147]. The SoK by Brandt, Filić, and Markelon [22] also proposes a KT scheme inspired by [28], [98] which only comprises an untrusted server and clients.

**Instantiation.** All real-world instantiations that we found additionally included third-party auditors.

## IV. Defining Authenticated Dictionaries

Here, we provide the API and security definitions for ADs.

### A. Bare-bones API

Recall that in ADs, the *server* is untrusted and may try to tamper with logs of previous states of the dictionary, or attempt to show divergent views to different users. Below, we define a bare-bones API for an AD, which we call a *stateful* AD to capture the notion that the server's database is mutating.

**Definition 1.** *A stateful AD, denoted* $\mathsf{AuthDS}$*, consists of a mutable parameter* $\mathsf{Epoch}$*, initialized to* $0$*, a set of permissible operations* $\mathsf{Ops} \subseteq \{\mathsf{insert}, \mathsf{delete}, \mathsf{update}\}$*, an update validity predicate* $F$*, and the following algorithms:*

- $\mathsf{pp} \xleftarrow{\$} \mathsf{AuthDS.Init}(1^\lambda)$ *takes as input a security parameter* $\lambda$ *and initializes* $\mathsf{AuthDS}$ *with public parameters* $\mathsf{pp}$*.*
- $(\mathsf{com}_1, \mathsf{state}_1)/\perp \xleftarrow{\$} \mathsf{AuthDS.Commit}\ (\mathsf{pp},\ D_{\mathsf{Init}})$ *takes as input the public parameters* $\mathsf{pp}$ *and a dictionary* $D_{\mathsf{Init}}$*. It outputs an internal cryptographic state* $\mathsf{state}_1$ *and a commitment* $\mathsf{com}_1$*. It also sets* $\mathsf{Epoch} = 1$*.*
- $(\mathsf{com}_{t+1}, \mathsf{state}_{t+1}, \pi_{t+1})/\perp \xleftarrow{\$} \mathsf{AuthDS.Update}(\mathsf{pp}, \mathsf{state}_t, \mathsf{updates})$ *takes as input the public parameters* $\mathsf{pp}$*, internal*

[2]Many works labeled "transparency" rely on external auditors; our "transparency" model covers only self-audited schemes without third-party auditors.

| Model | Roles | | | | | Example application | Deployment | Constructions |
|---|---|---|---|---|---|---|---|---|
| | Trusted data source | Client | PBB | Server | Auditor | | | |
| Private outsourced storage | | | | | – | Cloud storage, encrypted backups | No public deployments yet. | [70], [109], [122], [129], [130], [128], [117] |
| Public outsourced storage | | | | | – | Databases available on a cloud-hosted website | QLDB [8], immudb [84], [127] | [130], [70], [122], [129], [4], [37], [68], [133], [131] |
| Self-sovereign model | – | | | | | Key-transparency, news transparency | Keybase [86] | [82], [69] |
| Audited transparency | – | | | | | Key-transparency | WhatsApp [101], iMessage [6] | [28], [31], [54], [94], [97], [98], [100], [107] |
| Transparency | – | | | | – | Key-transparency | No public deployments yet. | [22], [144], [147] |

TABLE I

A SUMMARY OF OUR AD MODELS TAXONOMY. IN THIS TABLE, A RED COLORED CELL INDICATES THAT IN THE CORRESPONDING MODEL, THE RESPECTIVE PARTY IS "TRUSTED UNTIL CAUGHT" I.E., MISBEHAVIOR IS NOT PREVENTABLE BUT IS DETECTABLE. SIMILARLY, A YELLOW COLORED CELL INDICATES THAT A PARTY IS FUNGIBLE, I.E., A SINGLE FAILED INSTANCE OF THIS PARTY CAN BE REPLACED WITH ANOTHER INSTANCE. SYMBOL LEGEND: IMPLIES THIS PARTY RUNS THE LookupVerify OPERATION, INDICATES THAT *any* OPERATION ONLY VERIFIES IF THIS PARTY SIGNS IT, INDICATES THAT THIS PARTY IS RESPONSIBLE FOR SIGNING UPDATES TO A SUBSET OF THE DATA, DENOTES THAT THIS PARTY DOES NOT SIGN ITS UPDATE REQUESTS, DENOTES A SERVER THAT IS TRUSTED FOR AVAILABILITY, INDICATES THAT THIS PARTY IS TRUSTED FOR ACCESS CONTROL FOR UPDATE OPERATIONS, INDICATES THAT THIS PARTY AUDITS THE DATA STRUCTURE TO RELIEVE SOME OF THE COMPUTATIONAL BURDEN ON CLIENTS AND DENOTES THAT THIS PARTY OR MECHANISM IS RESPONSIBLE FOR DISBURSING A SMALL COMMITMENT TO CLIENTS.

*state* $\mathsf{state}_t$*, and a set of updates* $\{(\mathsf{op}_j, \mathsf{key}_j, \mathsf{value}_j)\}_j$ *where* $\mathsf{op}_j \in \mathsf{Ops}$*. This algorithm checks that* updates *is valid according to the validity predicate* $F$*; if not, it outputs* $\perp$*. If all checks pass, this algorithm returns a tuple* $(\mathsf{com}_{t+1}, \mathsf{state}_{t+1}, \pi_{t+1})$ *comprising the updated commitment, state, and proof of correct update.*

- $0/1 \leftarrow \mathsf{AuthDS.VerifyUpd}(\mathsf{pp}, \mathsf{com}_t, (\mathsf{com}_{t+i}, \pi_i)_{i=1}^n, t, n)$ *takes as input, public parameters* pp*, a set of simultaneous commitments* $\mathsf{com}_t, ..., \mathsf{com}_{t+n}$ *and proofs* $\pi_1, ..., \pi_n$*, an epoch* $t$*, and a number of epochs* $n$*. It verifies the proof* $\pi_i$ *with respect to* $\mathsf{com}_{t+i-1}$ *and* $\mathsf{com}_{t+i}$*, the other inputs and outputs* 0 *if the check fails and* 1*, if the check passes.*
- $(\mathsf{value}, \pi) \leftarrow \mathsf{AuthDS.Lookup}(\mathsf{state}_t, \mathsf{key})$ *takes as input an internal state* $\mathsf{state}_t$ *and a label* key*. If* key *is not included in the dictionary represented by* $\mathsf{state}_t$ *it outputs a proof of non-inclusion of* key *and* $\mathsf{value} = \perp$*, and otherwise it outputs the proof of inclusion of the tuple* $(\mathsf{key}, \mathsf{value})$*.*
- $0/1 \leftarrow \mathsf{AuthDS.LookupVerify}(\mathsf{pp}, \mathsf{com}, \mathsf{key}, \mathsf{value}, \pi)$ *verifies the proof* $\pi$ *that* com *includes* $(\mathsf{key}, \mathsf{value})$*.*

### B. Security definitions and additional APIs

The security definitions for data structures we aim to capture with AuthDS focus on binding the views of the values of a particular key at each timestep. Informally, this means that the values output by two different lookups for the same label should be equal, when verified with respect to the same commitment. This is formalized in Def. 4. In real applications, users may want to track changes made to values corresponding to particular keys, we formalize this next.

**Definition 2.** *A historical stateful AD is a stateful AD with the following additional algorithms:*

- $(\{(t_i, \mathsf{value}_i)_i\}, \pi) \leftarrow \mathsf{AuthDS.GetHistory}((\mathsf{state}_{t_i}, \mathsf{com}_{t_i})_i, \mathsf{key}, \{t_i\}_i)$*: takes as input a set of consecutive state and commitment tuples at the epochs* $t_i$ *for* $t_{i+1} = t_i + 1$ *and a key* key*. It outputs an (explicit or implicit) mapping* $(t_i, \mathsf{value}_i)_i$ *of the set of values associated with* key *at each epoch and the proof* $\pi$ *of its correctness.*
- $0/1 \leftarrow \mathsf{AuthDS.VerifyHistory}(\mathsf{pp}, \pi, (\mathsf{com}_{t_i})_i, \{(t_i, \mathsf{value}_i)\}_i)$*: parses its input and checks the proof* $\pi$ *that the mapping* $(t_i, \mathsf{value}_i)$ *verifies with respect to the sequence of commitments* $(\mathsf{com}_{t_i})_i$ *and the public parameters* pp *and outputs a bit.*

The main security definition for a historical authenticated dictionary is *history binding*. It begins with the standard assumption that all users can obtain identical commitments for each state or timestep of the dictionary via the public bulletin board. History binding then dictates the following: suppose Alice checks the history of a key $k$ over a time period $t_1$ through $t_n$. Based on the API in Def. 2, for any timestep $t$ between $t_1$ and $t_n$, Alice has an implicit notion of the value $v_t$ for $k$ at timestep $t$. If Bob later verifies a value $v_t^*$ for $k$ at timestep $t$, then $v_t$ and $v_t^*$ must match. To the best of our knowledge, the notion of history binding first appeared as the *soundness* property in [28], and Definition 3 matches [28] up to notational differences. This definition generalizes value binding by extending the *match* between two views of the same key over a longer time period.

**Oscillation attacks.** Without the history binding property, an adversarial server of an AD can mount what [112] termed an *oscillation attack*. The scenario where such an attack becomes relevant is when there are two kinds of parties for a given key $k$: one set of parties would like to track the changes to the values of $k$, let's call them the *monitors* of $k$, and another set of users would simply like to perform basic lookups on $k$'s value. The adversary is said to succeed if it can interleave updates or "oscillate" between versions of the directory such that it shows a history of values associated with $k$ to monitors

but a value $v$, that is not included in the monitors' history, to the client conducting the basic lookup for $k$. For example, a server that only responds to the monitors at odd epochs and to lookup clients at even ones could incorporate changes in even epochs that it immediately rolls back in odd ones.

**Generically compiling to support history.** Let $\mathsf{AuthDS}$ be an AD with states $(\mathsf{state}_i)_i$ and corresponding commitments $(\mathsf{com}_i)_i$. First, let us assume that all clients have the correct set of commitments $(\mathsf{com}_i)_i$ and that the clients are online at all times. In such a scenario, a client can retrieve a "history" of a key $\mathsf{key}$ between epochs $t, t+1, ..., t+n$ by requesting lookups at each epoch $t+i$. Denote this history $\{(t+i, \mathsf{value}_i)\}$ with corresponding lookup proofs $(\pi_i)_{i=0}^n$. Then, $\mathsf{AuthDS.VerifyHistory}$ can be implemented with proof $\Pi = (\pi_i)$ as the following: $\bigwedge_{i=0}^n \mathsf{AuthDS.LookupVerify}(\mathsf{pp}, \mathsf{com}_i, \mathsf{key}, \mathsf{value}_i, \pi_i)$. However, this sequence of commitments can be shuffled or have missing entries, for example, if there are network delays. Thus, we need a way to commit to the sequence of commitments so that intermediate states cannot be omitted. A simple solution is to update the algorithm $\mathsf{AuthDS.Update}$ to chain together the commitments by computing the newest commitment by hashing together the previous one, during each update. Now, if the commitment of two users diverges once, it must diverge forever. We discuss an example with static ADs in Sec. V-A.

**Shortcomings of label/history binding.** Value binding and history binding are purely syntactic: they ensure that any two query views for the same commitment agree, but do not guarantee that a read returns the most recent write. This distinction is intentional, as transparency models lack a trusted authority to authorize updates, while trusted-data-source settings allow update *correctness* to be verified cryptographically. This gap motivates a stronger guarantee–*read-write consistency*.

**Read-write consistency.** Here, we aim to capture two different models of updates: (1) an AD in which a single trusted data source authorizes all updates and (2) one in which each key is "owned" by a particular party who signs updates to this particular key. The following modifications to the update APIs of stateful ADs capture the ability of a particular party to verify an update with respect to a requested change for a key.

- $\pi/\perp \xleftarrow{\$} \mathsf{AuthDS.ProveSingleUpdate}(\mathsf{pp}, \mathsf{state}_t, \mathsf{state}_{t+1}, \mathsf{op}, \mathsf{key}, \mathsf{value})$ takes as input the public parameters $\mathsf{pp}$, two consecutive internal states $\mathsf{state}_t$ and $\mathsf{state}_{t+1}$, and an update $(\mathsf{op}, \mathsf{key}, \mathsf{value})$, where $\mathsf{op} \in \mathsf{Ops}$ and $(\mathsf{op}, \mathsf{key}, \mathsf{value}) \in \mathsf{updates}$ represents an update applied between $\mathsf{state}_t$ and $\mathsf{state}_{t+1}$. This algorithm outputs a proof $\pi$.
- $0/1 \leftarrow \mathsf{AuthDS.MonitorUpdate}(\mathsf{pp}, \mathsf{com}_t, \mathsf{com}_{t+1}, \mathsf{op}, \mathsf{key}, \mathsf{value}, \pi, t)$ takes as input the public parameters $\mathsf{pp}$, a pair of consecutive commitments $\mathsf{com}_t$ and $\mathsf{com}_{t+1}$, an update $(\mathsf{op}, \mathsf{key}, \mathsf{value})$, the proof $\pi$, and the epoch $t$. It verifies $\pi$ with respect to $\mathsf{com}_t$, $\mathsf{com}_{t+1}$, and the other inputs, outputting 1 if the check passes and 0 if it fails.

In self-sovereign settings, global audits offload the impractical task of each data source verifying system-wide invariants. For example, Keybase's monitor lets a user confirm their update [86], but without an append-only audit a malicious server could silently rollback valid changes.

Capturing the guarantee "no changes are made to a key" is nontrivial: our APIs only support retroactive checks to cover transparency scenarios, not preemptive enforcement. Instead, we require that if a client monitors an update $(\mathsf{op}, k, v)$ between epochs $t$ and $t+1$, then, assuming regular audits, any later lookup of $k$ must return either $v$ or a value $v'$ from another valid update $(\mathsf{op}', k, v')$ (i.e., one accompanied by a valid monitoring proof). We formalize this as *read-write consistency* in Definition 5.

## V. Variants of authenticated dictionaries

Now, we introduce a new categorization to systematize the literature on ADs. Solutions with a focus on privacy are relegated to Sec. V-C.

### A. Building blocks

Here, we list the building blocks and primitives most commonly used in the AD literature.

**Static ADs.** A *static AD* is an authenticated dictionary (Definition 1) without $\mathsf{Update}$ or $\mathsf{VerifyUpd}$—it commits to a single state $\mathsf{state}_t$ at epoch $t$ with commitment $\mathsf{com}_t^{\mathsf{static}}$ and supports only lookups. We found that a static AD explicitly or implicitly underlies all the AD constructions in the AD literature. To build an AD with updates, each update to the $t$-th epoch first computes a static AD commitment $\mathsf{com}_t^{\mathsf{static}}$ over the new state, then chains it with the previous commitment (e.g., $\mathsf{H}(\mathsf{com}_{t-1} \| \mathsf{com}_t^{\mathsf{static}})$) to get the full commitment $\mathsf{com}_t$.

**Merkle trees.** Merkle trees (MTs) hash a set of leaf values into a root by recursively hashing child nodes. Variants that support updates include the following. A *chronological Merkle-tree* (CMT) refers to a binary MT which augments a simple MT with insertions in purely chronological order. A *Lexicographic Merkle tree* (LMT) is a hash tree that is searchable by labels. *Patricia Merkle Trees* (PMTs) and *Sparse Merkle Trees* (SMTs) reduce storage for LMTs via path compression and implicit subtrees. See full version for further discussion.

**Skip lists.** A skip list is a probabilistic data structure that stores a set of elements $S$ from an ordered universe as a series of linked lists. For a sequence of size $n$, skip lists support search and insertion operations with an average time complexity of $\mathcal{O}(\log n)$. See full version for more details.

**Algebraic tools for ADs.** Algebraic solutions use group exponentiation under RSA or pairing assumptions to commit to sets or multisets. Without data structure optimizations, computing membership witnesses in both RSA-based and pairing-based constructions takes $\mathcal{O}(N)$ time. See full version for details.

### B. Indexed data structures

Building on the *static AD* snapshot at each epoch (see above), indexed ADs organize keys for efficient lookups and proofs. We classify them into three types:

1) Mutating ADs: the static AD at epoch $t$ stores only each key's latest value [113], [10], [147], [146], [92].

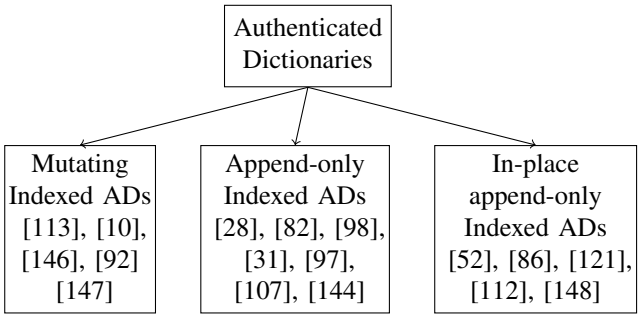



Fig. 2. A taxonomy of authenticated dictionary solutions.

2) Append-only ADs: the static AD at epoch $t$ retains all values ever inserted for each key, in insertion order [28], [107], [98], [31], [97], [82], [144].
3) In-place append-only ADs: the static AD at epoch $t$ maps each key to its full history as the current "value" [86], [121], [112], [148], [52].

Each of these construction types ultimately achieves similar asymptotic performance—$\log n$ time for both lookups and updates—except for a few algebraic variants that offer constant-time updates at the cost of linear-time lookups. The primary difference between categories lies in the efficiency of history checking. This observation is borne out of Table II and further illustrated in the detailed analysis below.

**Mutating ADs.** ARPKI [10] uses a lexicographic Merkle tree whose leaf label is $\mathsf{H}(k)$ and leaf value is the current record for $k$. In ARPKI's application, the keys are domain names and the value for a key is a compilation of certificates for the domain, signed by multiple authorities. Verkle trees [92] generalize this to higher-arity trees via pairings, shaving the Lookup proof length by a constant factor. Algebraic constructions reduce the usual linear per-proof cost via batching: UAD [146] (RSA-based) aggregates lookup proofs across multiple keys using batching techniques [18], [143]. VeRSA's [147] mutating AD adds per-key version numbers, which are incremented on each update, to detect unauthorized changes without third-party auditors, while still amortizing proof generation.

**Append-only ADs.** We defer a discussion of most of the constructions in this category to Sec. V-C, and focus here on two constructions. Tomescu et al.'s AAD [144] uses a bilinear-pairing-based *append-only authenticated set* (AAS) for each key's values. Their AAS combines prefix-trees, pairing proofs for (non-)membership, and a logarithmic "forest" of disjoint trees that are merged as they grow. This yields $O(\log N)$ audit proofs and verification, but updates can cost $O(N)$ in the worst case when trees merge. Hu et al.'s Merkle$^2$ [82] also builds a forest of chronological Merkle trees whose roots commit to prefix-trees of child logs. Lookup scans all subtrees for a key, and while most operations match the performance of the constructions discussed in Sec. V-C, updates and audits remain $O(N)$ in the worst case due to full-forest merges.

**In-place append-only ADs.** These schemes store each key's full history as the leaf value of a lexicographic Merkle tree (LMT). Keybase uses per-user signed sigchains chained via a linked list [86]; Mog replaces the linked list with a chronological Merkle tree for faster lookups [112]; Chainiac adds skipchains to enable efficient monitoring over arbitrary intervals [121]; Fahl et al. [52] run a CMT alongside a balanced 2–3 tree LMT ("FixTree") for smaller proofs; and Verdict [148] commits sorted label-value linked lists in fixed-depth Merkle trees, doubling capacity by adding new trees as needed to maintain a constant leaf depth.

*1) SNARK-aided solutions:* In our discussion of ADs so far, the total cost for verifying updates (Audit) is still linear in the number of epochs over which the audit is being conducted. This cost is impractical for smaller clients who do not trust third-party auditors. SNARKs [14], which allow short and efficiently verifiable proofs of computational statements, can help. Essentially, a SNARK is used to generate a proof that a commitment $\mathsf{com}'$ was obtained by applying a sequence of valid updates to an AD committed in com. However, most efficient SNARKs require a pre-processed common reference string (CRS) [17], which pre-determines the instance a verifier can verify. Thus, ADs whose audit proof sizes vary depending on inputs are less amenable to SNARK-based solutions.

Verdict [148] uses a SNARK-friendly data structure (with fixed depth), which we discussed in Sec. V-B to support SNARK-based auditing. They also customize existing SNARK techniques for efficient audits using incrementally verifiable computation (IVC) [149] to "incrementally" prove the correctness of updates over a sequence of epochs. VeRSA's [147] two RSA-based constructions also use SNARK-based techniques to enable client auditing to verify the invariant that version numbers are always incrementing. The first construction, VeRSA-IVC, implements Audit for a small number of updates by embedding their RSA-based AD in a SNARK. Combined with IVC techniques, this allows efficient Audit with verifier time independent of the number of epochs. Their other construction, VeRSA-amtz, doesn't use IVC, which may sometimes be infeasible, particularly since different users may want to run Audit with different start and end epochs; instead, it relies on a skip-list-like technique.

### C. Privacy in AD implementations

We now discuss the most well-studied aspect of privacy in ADs: privacy for particular keys in the dictionary. This notion of privacy is only meaningful when there is no single trusted data source that owns all keys, and is therefore primarily relevant in transparency-related settings. Other privacy issues arising in AD-based systems are discussed in the appendix of the full version. Recall, in our model, the server is trusted to store the entire database and to enforce access control over which clients may query a given key. In applications such as CT, however, any user can typically query any key. Thus, privacy for individual keys has been studied mainly in the context of KT (see the example application in Sec. III), where users may not wish to publicly reveal their usernames or public keys. We distinguish between two forms of key-related privacy: *content hiding* and *metadata hiding*.

| Protocol | Added Assum. | Motivating Application | Lookup operations | | History operations | | Update | Verify Upd./Audit |
|---|---|---|---|---|---|---|---|---|
| | | | Client | Server | Client | Server | | |
| SEEMless [28] | DL | Keys | $\log N^*$ | $\log N^*$ | $(u + \log T) \log N$ | $(u + \log T) \log N$ | $k \log(N + k)^*$ | $k \log(N + k)^*$ |
| Parakeet [107] | DL | Keys | $\log N^*$ | $\log N^*$ | $(u + \log T) \log N$ | $(u + \log T) \log N$ | $k \log(N + k)^*$ | $k \log(N + k)^*$ |
| OPTIKS [98] | DL | Keys | $u \log N^*$ | $u \log N^*$ | $u \log N$ | $u \log N$ | $k \log(N + k)^*$ | $k \log(N + k)^*$ |
| ELEKTRA [97] | DDH | Keys | $(u \log N + \log T)^*$ | $(u \log N + \log T)^*$ | $(u \log N + \log T)^*$ | $(u \log N + \log T)^*$ | $(k \log(N + k) + \log T)^*$ | $(k \log(N + k) + \log T)^*$ |
| RZKS [31] | DDH | Generic | $(\log N + \log T)^*$ | $(\log N + \log T)^*$ | N/A | N/A | $(k \log(N + k) + \log T)^*$ | $(k \log(N + k) + \log T)^*$ |
| Merkle$^2$ [82] | - | Generic | $\log N^2$ | $\log N^2$ | $\log N^2$ | $\log N^2$ | $\log N^2 *$ | $\log N^2 *$ |
| AAD [144] | q-SBDH | Generic | $\log N^2$ | $\log N^2$ | Naive | Naive | $\log N^3 *$ | $\log N^*$ |
| Balloon [133] | PKI | Transparency | $\log N$ | $\log N$ | $u \log N$ | $u \log N$ | $\log N$ | $\log N$ |
| UAD [146] | sRSA | Generic | 1 | $(\lambda N \log N/m)^{**}$ | Naive | Naive | $k \log k$ | 1 |
| VeRSA-IVC [147] | q-SBDH, sRSA | Generic | 1 | $(N/m + \log m)^*$ | $u$ | $(N/m + \log m)^*$ | $k$ | $\log T$ |
| VeRSA-amtz [147] | q-SBDH, sRSA | Generic | 1 | $(N/m + \log m)^*$ | $u$ | $(N/m + \log m)^*$ | $k \log T^*$ | $\log T$ |
| ARPKI [10]$^\dagger$ | - | Certificates | $\log N$ | $\log N$ | $T \log N$ | $T \log N$ | $k \log(n + k)$ | $k \log(n + k)$ |
| CONIKS [113] | DL | Keys | $\log N^*$ | $\log N^*$ | Naive | Naive | $k \log(n + k)^*$ | $k \log(n + k)^*$ |
| AT [52]$^\dagger$ | - | Software | $\log N$ | $\log N$ | $\log N + u$ | $\log N + u$ | $k + \log(n + k)$ | $k + \log(n + k)$ |
| Verkle trees [92]$^\S$ | q-SBDH | Generic | $\log_v(N)$ | $\log_v(N)$ | Naive | Naive | $kv \log_v(N)$ | $kv \log_v(N)$ |
| AD-skip-list [70] | PKI | Generic | $\log N^*$ | $\log N$ | Naive | Naive | $\log N^*$ | $\log N^*$ |
| PAD-rb-tree [4] | PKI | CRL | $\log N$ | $\log N$ | $u \log n$ | $u \log n$ | $\log n$ | $\log n$ |
| PAD-skip-list [4] | PKI | CRL | $\log N^*$ | $\log N^*$ | $u \log n^*$ | $u \log n^*$ | $\log n^*$ | $\log n^*$ |
| Chainiac [121]$^\dagger$ | - | Software | $\log u + \log n$ | $\log u + \log n$ | $u + \log n$ | $\log n$ | $k \log u \log n$ | $k \log u \log n$ |
| Mog [112]$^\dagger$ | - | Generic | $\log u + \log n$ | $\log u + \log n$ | $u + \log n$ | $\log n$ | $k \log u \log n$ | $k \log u \log n$ |
| Keybase [86]$^\dagger$ | - | Generic | $u + \log n$ | $\log n$ | $u + \log n$ | $\log n$ | $k \log n$ | $k \log n$ |
| Verdict [148]$^\dagger$ | SXDH | Generic | $u + \log n$ | $\log n$ | $u + \log n$ | $\log n$ | $k \log n$ | $\log(k \log n)/T$ |

TABLE II

EFFICIENCY COMPARISON OF VARIOUS AUTHENTICATED DICTIONARY CONSTRUCTIONS FOR DIFFERENT APPLICATIONS. GetHistory AND VerifyHistory ARE NOT EXPLICITLY SUPPORTED IN SOME WORKS, SO "NAIVE" INDICATES THAT A HISTORY CHECK EQUATES TO A LOOKUP AT EVERY EPOCH. FOR RZKS [31], NO KEY UPDATES ARE ALLOWED, AND AUDITS ENSURE NO ENTRY REMOVAL, MAKING HISTORY CHECKS IRRELEVANT; A SINGLE LOOKUP SUFFICES TO KNOW THE KEY'S HISTORY IN RZKS' AD. THE TABLE NOTATION IS AS FOLLOWS: $\lambda$ IS THE SECURITY PARAMETER, $N$ IS THE NUMBER OF EXISTING ITEMS IN THE CORE DATA STRUCTURE, $k$ IS THE NUMBER OF ELEMENTS ADDED PER EPOCH, $T$ IS THE NUMBER OF SERVER EPOCHS SINCE THE LAST CALL TO THIS OPERATION, AND $u$ IS THE NUMBER OF UPDATES MADE FOR THE VALUE OF A KEY. COMPLEXITIES WITH $*$ INDICATE AVERAGE CASES, WITH $m$ BEING THE NUMBER OF OPERATIONS IN A BATCH (WHERE APPLICABLE). FOR CONSTRUCTIONS MARKED WITH $^\dagger$, $n$ IS THE NUMBER OF KEYS, SO $N = un$. CONSTRUCTIONS MARKED WITH $^\ddagger$ SUPPORT NO KEY VALUE MUTATION. VERKLE TREES, MARKED $^\S$, INCLUDE PARAMETER $v$ FOR VECTOR COMMITMENT CAPACITY.

**Content hiding.** Content hiding protects keys and values from unauthorized parties (e.g., other users or auditors in a KT system). Standard Merkle-tree proofs reveal sibling paths, leaking extra information. CONIKS [113] addresses this issue in an ARPKI [10]-like solution by deriving leaf labels via a verifiable random function (VRF), so only the server (with the VRF secret) can map keys to labels. Clients verify VRF outputs with the public key, preventing attackers from using rainbow tables to recover queried keys.

**Metadata hiding.** Metadata hiding conceals when and how often a key is updated or accessed. CONIKS [113] masks keys with VRFs but still leaks metadata via sibling proofs (the "tracing vulnerability"). SEEMless [28] introduces a zero-knowledge-with-leakage definition using PMTs+VRFs and an "expired" tree of stale labels to prove freshness without revealing update history. Follow-up works enhance this design by improving storage efficiency [107], reducing history-verification cost with signed updates [98], [97], adding post-compromise VRF security [31], and supporting deletion and account transfer [107], [98].

## VI. Concluding discussion

### A. Performance barrier

In this section, we discuss the performance barrier faced by implementers of AD constructions.

**Non-barriers.** Thus far, the systems literature has shown that once a particular asymptotic performance regime is chosen, practitioners are able to make significant progress on systems-level performance and scalability, as illustrated, for example, by [107], [98], [45], [102]. Hence, we believe that the asymptotic performance is the more interesting criterion when considering constructions. Among asymptotic metrics of AD performance, storage complexity seems to be a solved problem, as most constructions (all from the last 10 years) have storage overhead linear in the number of keys, with an additional linear overhead for storing history. Thus, asymptotically, adding authentication to a database does not increase its storage overhead. Parakeet [107] and OPTIKS [98] have discussed modifying security definitions to support garbage collection, hence keeping systems practical long-term.

For the time complexity of basic AD operations, client-side verification operations and proof size can both be reduced to constant using generic (and rapidly improving) cryptographic tools such as SNARKs (see e.g., [147]), albeit at the cost of increased server compute overhead.

**Gap in server-side operation asymptotics.** As discussed above, there already exist constructions and techniques that allow ADs to match the asymptotic storage, verification time, and bandwidth of non-authenticated counterparts. On the other hand, as Table II shows, the server-side time complexity of AD constructions bifurcates as follows: $\mathcal{O}(\log n)$ for both lookups and updates, or constant overhead for one operation but linear in $n$ for the other. We have noted previously (see Sec. II-B) that an AD can be used to instantiate a memory checker. Thus, the lower bounds on memory checkers apply to ADs. Memory checker performance is widely measured in terms of query complexity: the number of distinct memory locations accessed by the server in order to prove the correctness of one logical operation (i.e. a lookup or an update). Assuming that each logical memory access requires at least constant time gives us a lower bound on the time complexities of lookups and updates. Let $q_r$ denote read query complexity and $q_w$ denote write query complexity. Recently, Boyle et al. [21] showed, among other things, that if either $q_r$ or $q_w$ is (1) $\mathrm{o}(\log n/\log\log n)$, then the other query complexity must be $(\log n)^{\omega(1)}$, or (2) $\mathcal{O}(1)$, then the other operation must be $n^{\Omega(1)}$. They also give constructions showing that this lower bound is tight in terms of query complexity. There exist AD constructions (e.g. [147], [146]) that meet the time complexity lower bound for the case where one operation is constant time but the other is linear.

The following problem remains open:

> *Does there exist an AD construction that simultaneously achieves time complexity (1) $o(\log N)$ for a* Lookup*, and (2) $\mathcal{O}(\log N)$ (or smaller) for an* Update*?*

### B. Comparison and Cross-compilation

As presented in Sec. III-B, the different AD models feature distinct settings and/or trust assumptions. Fortunately, many AD schemes can be adapted to work in a different model.

**Translations without added cryptographic assumptions.** In Fig. 1, arrows between models indicate compilations that do not require additional cryptographic assumptions. For example, schemes in the Transparency model also apply to the Audited Transparency model, as the auditor(s) can simply be ignored. Likewise, the Self-Sovereign model can be adapted to the Private and Public Outsourced Storage models by adjusting participant roles: in the Private model, auditor(s) remain idle and a single client acts as the data source; in the Public model, auditors are also idle and one client serves as the data source while the other clients may query. Finally, the Public Outsourced Storage model compiles into the Private model by allowing the data source to also be the client. Other translations require additional cryptographic assumptions, discussed below.

**Introducing a trusted data source.** The gap between models with and without a trusted data source is larger than that between models within the same category, as shown in Fig. 1. Models with a trusted data source rely on stronger assumptions, i.e., the existence of an honest party. This party often holds secrets and can be authenticated via a PKI, making schemes in this category difficult to adapt to settings without such trust or infrastructure. In contrast, schemes from the transparency models (Fig. 1b) can be adapted to trusted data source settings by introducing (i) a trusted party to enforce semantic correctness and (ii) primitives that allow identification of the trusted party. However, achieving stronger security guarantees requires additional overhead.

Due to the trusted data source model's inherently stronger assumptions, it is not possible to compile trusted data source solutions in a black-box manner to the transparency models.

**Other translations.** Ghosh and Chase [62] show how to transform a scheme in the Audited Transparency model (specifically OPTIKS [98]) into an auditor-free scheme with minimal changes to the data structures. This transformation comes at the cost of weaker security: while the original scheme ensures strong consistency (a user will detect a key that was forged while they were offline as soon as they return online), the auditor-free version offers only weak consistency (if a fake key is issued while the user is online, either the user or recipient will detect it the next time they are both online).

To translate schemes from the Private Outsourced Storage Model to either the Self-Sovereign or Public Outsourced Storage models we require PKI so that clients can verify authenticity, e.g., using digital signatures; the client logic should also be updated to include both verification of the signature and inclusion/exclusion proofs. Both settings additionally require multiple clients. In the Self-Sovereign model, some of these clients may be the data source and in the Public Outsourced Storage model, an additional distinct data source is required.

### C. Comparing security definitions and assumptions

So far in the literature, read-write consistency has only been considered in the memory checking setting, which is equivalent to our setting with a single trusted data source. We are the first to formalize read-write consistency when multiple distinct parties may be trusted writers for particular keys in an AD. This read-write consistency definition is the strongest we have considered and (where applicable) implies other properties as follows: Read-write consistency $\implies$ History-binding $\implies$ Value-binding.

However, achieving read-write consistency requires each writer (or client) to manage cryptographic keys robustly, which conflicts with the design goals of transparency systems. These systems deliberately eschew per-user key management in order to simplify deployment and user experience. Consequently, in fully decentralized or "no trusted source" scenarios, *history-binding* emerges as the most appropriate security notion: it prevents equivocation over time without imposing onerous key-management burdens on end users.

If transparency evolves to incorporate a trusted data source (i.e., by introducing signatures), a solution initially satisfying history-binding or value-binding can be upgraded to achieve read-write consistency. That said, transparency-oriented schemes can be upgraded to full read-write consistency by incorporating signatures from a trusted data source (or multiple sources). This can be achieved using the compiler from the work of Falzon et al. [53], which adds a requirement to a transparency construction to ensure that the number of dictionary changes aligns with expectations following an update. Fortunately, update proofs generated by the constructions and techniques detailed in Sec. V satisfy this requirement because they implicitly encode the count of changes between dictionary versions, facilitating straightforward rollback detection. This raises the following open problems:

*Can we design a compiler that transforms any AD satisfying history-or value-binding into one with full read-write consistency, without introducing new requirements as [53] does, or asymptotic overhead?*
*Do all AD constructions that achieve value- or history-binding inherently produce update proofs that commit to the exact count of changes between versions (thus enabling rollback detection)?*

### D. Auditing assumptions

Many works assume third-party auditors verify all updates to the AD to lower client costs, but this generally requires auditors to run algorithms linear in the number of inserted items. This raises several persistent questions [105]. Who will perform this non-trivial auditing task and how will they be incentivized? Incentives are tricky as ideally multiple auditors would perform the same task (for robustness) but it is difficult to prove that they have done so as in nearly all practical cases auditing will fail to turn up any misbehavior. Most current works leave this question open, hoping that for high-value applications auditors might be corporations, hobbyists or non-profits, similar to the parties running CT logs today.

Some works instead aim for the stronger goal of *client-auditing* or *self-auditing*, in which auditing is cheap enough (sublinear in the size of the dictionary) so ordinary clients can verify updates themselves. One method uses SNARKs to succinctly prove correct AD updates [33], [136], though they impose high server-side costs. VeRSA [147] achieves client-auditability without generic SNARKs by leveraging an algebraic AD construction for more efficient proofs.
Recently, Ghosh and Chase [62] proposed an alternative model with a weaker security guarantee, applicable only if a lookup for a key is between two checks by the key owner. This particular model increases lookup and update request costs but also eliminates the need for third-party auditors.
Brorsson et al. [23] propose an idea where users sign off on the AD's commitments in key-transparency, similar to blockchain staking solutions such as Algorand [64]. This could potentially be adapted for auditing, but needs further study.

This leaves us with several open problems:

*How can we incentivize auditing in practice?*
*Can we achieve acceptable practical performance replacing auditing using generic SNARKs?*
*Can we formalize the relationships between third-party auditing and newer models such as VeRSA's?*

### E. Practical and Regulatory Considerations

**Reputational risks and rewards.** As of this writing, both the actual benefit of deploying ADs as well as the benefit perceived by users remain open questions. There is not yet clear evidence of a large upside for applications deploying authenticated dictionaries. A recent study published in PETS [56] investigated user perceptions of key transparency in WhatsApp and found that ordinary users generally lack

sufficient knowledge to meaningfully increase trust *perception* in the service—though this may change over time. In the case of trusted data source implementations, such as enterprise databases, we found only a handful of deployments [84], [8] (of which QLDB is currently being phased out) and no user studies to guide further development.

On the other hand, even an honest server might have issues due to bugs or random bit-flips [9], leading to a clear reputational risk. Various works [31], [98] consider one kind of recovery—recovery from an attack that compromises users' privacy—but not recovery in case any of the verification operations LookupVerify, VerifyHistory, or Audit fail.

*More work is needed to (1) further user outreach and follow up with user studies to evaluate the reputational benefit of deploying ADs, and (2) define new threat models that account for reputational recovery from benign errors such as random bit-flips or temporary compromise, while ensuring that genuine cheating does not go unpunished.*

**Regulatory compliance issues.** Recent privacy and anti-trust regulation, particularly in Europe, might be in conflict with adding transparency to certain applications. For example, in the context of key transparency, user data is subject to strict privacy requirements, and a privacy breach–such as the one demonstrated by Kondracki et al. [87] for CT logs, where previously hidden domains were exposed–would be unacceptable. Academic work such as that of Len et al. [99] explored the question of interoperability for ADs in the transparency setting, addressing the recent Digital Markets Act (DMA) [51], which requires encrypted messaging systems to provide interoperable messaging across application providers. Recently, Garg and Elahi [59] provided a solution for interoperable key-transparency. Parakeet [107] and OPTIKS [98] discuss deletion models that allow for a form of valid deletion, addressing the "right to be forgotten" [60] (which is seemingly incompatible with history checking) included in the European Commission's GDPR regulation. Future research may explore similar questions in other authenticated dictionary models, particularly when handling large-scale user data.

## VII. Conclusion

Authenticated dictionaries are now at the core of important systems such as end-to-end encrypted messaging, software transparency, verifiable cloud storage, and blockchains, but have evolved separately for different applications with incompatible roles, APIs, and threat models. This work brings those lines together, unifying five deployment models, reconciling the main security definitions (value binding, history binding, and read–write consistency), and organizing more than thirty constructions into a common taxonomy with clear performance tradeoffs. In doing so, we exposed a persistent barrier: known designs either pay $\mathcal{O}(\log n)$ time for both lookups and updates, or drive one operation to $\mathcal{O}(1)$ only by letting the other degrade to linear time. This holds true even when we strengthen trust assumptions, challenging the intuition that "just trust someone" will automatically improve performance.

We intend this work not as the last word but as infrastructure for future research; a shared vocabulary for trust assumptions, APIs, and security goals should support modular constructions, reusable AD components, fair comparison of systems, and reasoning about the guarantees actually needed in practice. We also highlight several concrete research directions. In particular, we ask if it is possible to design realistic auditing and incentive mechanisms so that existing applications without integrity guarantees can adopt ADs at all. Answering these questions is what will turn ADs into standard security plumbing for the next generation of large-scale systems.

## Acknowledgments

The authors would like to thank Emanuel Opel for his early contributions to the modeling of authenticated dictionaries, and Esha Ghosh and Andrew Miller for helpful discussions. Francesca Falzon was supported by Armasuisse Science and Technology. Joseph Bonneau was supported by a16z crypto, NSF grant CNS-2239975, DARPA Agreement HR00112020022, and gifts from Google, Meta, and Handshake Labs. Harjasleen Malvai was supported in part by NSF award #1943499. The views and conclusions contained in this material are those of the authors and do not necessarily reflect the official policies or endorsements of the US Government, DARPA, a16z Crypto, or any other supporting organization.

## References

[1] P. AG. Proton AG. https://proton.me/, 2025.

[2] M. Al-Bassam and S. Meiklejohn. Contour: A practical system for binary transparency. In *ESORICS*, 2018.

[3] M. AlSabah, A. Tomescu, I. Lebedev, D. Serpanos, and S. Devadas. Privipk: Certificate-less and secure email communication. *Computers & Security*, 70, 2017.

[4] A. Anagnostopoulos, M. T. Goodrich, and R. Tamassia. Persistent authenticated dictionaries and their applications. In *ISC*, 2001.

[5] Apple's certificate transparency log program. https://support.apple.com/en-by/103703.

[6] Apple Inc. imessage contact key verification. https://security.apple.com/blog/imessage-contact-key-verification/, 2023. Accessed: 2025-07-21.

[7] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song. Provable data possession at untrusted stores. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 598–609. ACM, 2007.

[8] A. AWS. Data verification in Amazon QLDB. https://docs.aws.amazon.com/qldb/latest/developerguide/verification.html#verification.how-it-works.proof, 2025.

[9] A. Ayer. How Certificate Transparency Logs Fail and Why It's OK. www.agwa.name/blog/post/how_ct_logs_fail.

[10] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski. Arpki: Attack resilient public-key infrastructure. In *ACM CCS*, 2014.

[11] D. Bayer, S. Haber, and W. S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II: Methods in Communication, Security, and Computer Science*, 1993.

[12] J. Benaloh and M. de Mare. Efficient broadcast time-stamping. *Technical Report 1 TR-MCS-91-1 Clarkson University Department of Mathematics and Computer Science*, 1991.

[13] J. Benaloh and M. De Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Eurocrypt*, 1994.

[14] N. Bitansky, R. Canetti, A. Chiesa, S. Goldwasser, H. Lin, A. Rubinstein, and E. Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4), 2017.

[15] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 90–99, 1991.

[16] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12:225–244, 1994.

[17] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *STOC*, 1988.

[18] D. Boneh, B. Bünz, and B. Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In *CRYPTO*, 2019.

[19] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *Asiacrypt*, 2001.

[20] J. Bonneau. Ethiks: Using ethereum to audit a coniks key transparency log. In *Financial Crypto*, 2016.

[21] E. Boyle, I. Komargodski, and N. Vafa. Memory checking requires logarithmic overhead. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1712–1723, 2024.

[22] N. Brandt, M. Filić, and S. A. Markelon. SoK: On the security goals of key transparency systems. Cryptology ePrint Archive, Paper 2024/1938, 2024.

[23] J. Brorsson, E. Pagnin, B. David, and P. S. Wagner. Consistency-or-die: Consistency for key transparency. *Cryptology ePrint Archive*, 2024.

[24] A. Buldas and P. Laud. New linking schemes for digital time-stamping. In *ICISC*, volume 98. Citeseer, 1998.

[25] A. Buldas, H. Lipmaa, and B. Schoenmakers. Optimally efficient accountable time-stamping. In *PKC*, 2000.

[26] V. Buterin. Having a safe cex: proof of solvency and beyond. https://vitalik.ca/general/2022/11/19/proof_of_solvency.html.

[27] D. Catalano and D. Fiore. Vector commitments and their applications. In *Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16*, pages 55–72. Springer, 2013.

[28] M. Chase, A. Deshpande, E. Ghosh, and H. Malvai. Seemless: Secure end-to-end encrypted messaging with less trust. In *ACM CCS*, 2019.

[29] M. Chase and S. Meiklejohn. Transparency overlays and applications. In *ACM CCS*, 2016.

[30] P. Chatzigiannis, F. Baldimtsi, and K. Chalkias. Sok: Blockchain light clients. In *International Conference on Financial Cryptography and Data Security*, pages 615–641. Springer, 2022.

[31] B. Chen, Y. Dodis, E. Ghosh, E. Goldin, B. Kesavan, A. Marcedone, and M. E. Mou. Rotatable zero knowledge sets: Post compromise secure auditable dictionaries with application to key transparency. In *Asiacrypt*, 2023.

[32] J. Chen, H. W. Lim, S. Ling, H. Wang, and H. Wee. Shorter ibe and signatures via asymmetric pairings. In *Pairing Workshop*, 2013.

[33] W. Chen, A. Chiesa, E. Dauterman, and N. P. Ward. Reducing Participation Costs via Incremental Verification for Ledger Systems. Cryptology ePrint Archive, Paper 2020/1522, 2020.

[34] A. Chepurnoy, C. Papamanthou, S. Srinivasan, and Y. Zhang. Edrax: A cryptocurrency with stateless transaction validation. *Cryptology ePrint Archive*, 2018.

[35] M. Christ and J. Bonneau. Limits on revocable proof systems, with implications for stateless blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 54–71. Springer, 2023.

[36] L. Chuat, P. Szalachowski, A. Perrig, B. Laurie, and E. Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *IEEE CNS*, 2015.

[37] S. A. Crosby and D. S. Wallach. Authenticated dictionaries: Real-world costs and trade-offs. *ACM Trans. Inf. Syst. Secur.*, 14(2), Sept. 2011.

[38] G. G. Dagher, B. Bünz, J. Bonneau, J. Clark, and D. Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 720–731, New York, NY, USA, 2015. Association for Computing Machinery.

[39] R. Dahlberg and T. Pulls. Verifiable light-weight monitoring for certificate transparency logs. In N. Gruschka, editor, *Secure IT Systems*, pages 171–183, Cham, 2018. Springer International Publishing.

[40] R. Dahlberg and T. Pulls. Verifiable light-weight monitoring for certificate transparency logs. In *NordSec*, 2018.

[41] R. Dahlberg, T. Pulls, T. Ritter, and P. Syverson. Privacy-preserving & incrementally-deployable support for certificate transparency in tor. *PETS*, 2021(2), 2021.

[42] R. Dahlberg, T. Pulls, J. Vestin, T. Høiland-Jørgensen, and A. Kassler. Aggregation-based gossip for certificate transparency. corr abs/1806.08817 (2018). *arXiv preprint arXiv:1806.08817*, 2018.

[43] I. Damgård and N. Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Paper 2008/538, 2008.

[44] A. Delignat-Lavaud, C. Fournet, K. Vaswani, M. Costa, S. Clebsch, and C. M. Wintersteiger. Transparent attested dns for confidential computing services. *arXiv preprint arXiv:2503.14611*, 2025.

[45] Y. Deng, M. Yan, and B. Tang. Accelerating merkle patricia trie with gpu. *Proceedings of the VLDB Endowment*, 17(8):1856–1869, 2024.

[46] B. Dowling, F. Günther, U. Herath, and D. Stebila. Secure logging schemes and certificate transparency. In *European Symposium on Research in Computer Security*, pages 140–158. Springer, 2016.

[47] A. Dutta, S. Bagad, and S. Vijayakumaran. Mprove+: Privacy enhancing proof of reserves protocol for monero. *IEEE Transactions on Information Forensics and Security*, 16, 2021.

[48] A. Dutta and S. Vijayakumaran. Mprove: A proof of reserves protocol for monero exchanges. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 330–339, 2019.

[49] C. Ellison and B. Schneier. Ten risks of PKI: What you're not being told about public key infrastructure. *Computer Security Journal*, 16(1), 2000.

[50] S. Eskandarian, E. Messeri, J. Bonneau, and D. Boneh. Certificate transparency with privacy. *PETS*, 2017.

[51] EU Digital Markets Act (DMA). eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32022R1925, 2022.

[52] S. Fahl, S. Dechand, H. Perl, F. Fischer, J. Smrcek, and M. Smith. Hey, nsa: Stay away from my market! future proofing app markets against powerful attackers. In *ACM CCS*, 2014.

[53] F. Falzon, H. Malvai, and E. Opel. Black-box approaches to authenticated dictionaries: New constructions and lower bounds. *Cryptology ePrint Archive*, 2025.

[54] V. Fang, E. Dauterman, A. Ravoor, A. Dewan, and R. A. Popa. Legolog: A configurable transparency log. In *10th IEEE European Symposium on Security and Privacy, EuroS&P 2025, Venice, Italy, June 30 - July 4, 2025*. IEEE, 2025.

[55] L. Ferran and R. Schwartz. Cyber spy program flame compromises key microsoft security system. *ABC News*, June 2012.

[56] K. Fischer, M. Keil, A. Buckmann, and M. A. Sasse. " if you want to encrypt it really, really hardcore...": User perceptions of key transparency in whatsapp. *Proceedings on Privacy Enhancing Technologies*, 2025.

[57] E. Galperin, S. Schoen, and P. Eckersley. A Post Mortem on the Iranian DigiNotar Attack. *EFF DeepLinks Blog*, 2011.

[58] A. Garba, A. Bochem, and B. Leiding. Blockvoke–fast, blockchain-based certificate revocation for pkis and the web of trust. In *ISC*, 2020.

[59] N. Garg and M. T. Elahi. (p) kt-iee: Secure key transparency protocols for interoperable end-to-end encrypted message systems. In *Free and Open Communications on the Internet 2024: FOCI 2024*, pages 68–76. Privacy Enhancing Technologies Board, 2024.

[60] GDPR Right to be Forgotten. gdpr-info.eu/issues/right-to-be-forgotten/.

[61] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In T. Johansson and P. Q. Nguyen, editors, *Eurocrypt*, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[62] E. Ghosh and M. Chase. Weak consistency mode in key transparency: Optiks. Cryptology ePrint Archive, Paper 2024/796, 2024. https://eprint.iacr.org/2024/796.

[63] E. Ghosh, O. Ohrimenko, D. Papadopoulos, R. Tamassia, and N. Triandopoulos. Zero-knowledge accumulators and set algebra. In *Asiacrypt*, 2016.

[64] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

[65] S. Goldberg, L. Reyzin, D. Papadopoulos, and J. Včelák. Verifiable Random Functions (VRFs). RFC 9381, 2023.

[66] D. Goodin. State-sponsored hackers in china compromise certificate authority. *Ars Technica*, November 2023.

[67] M. T. Goodrich, E. M. Kornaropoulos, M. Mitzenmacher, and R. Tamassia. Auditable data structures. In *IEEE EuroS&P*. IEEE, 2017.

[68] M. T. Goodrich, C. Papamanthou, and R. Tamassia. On the cost of persistence and authentication in skip lists. In *Experimental Algorithms: 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007. Proceedings 6*, 2007.

[69] M. T. Goodrich, M. Shin, R. Tamassia, and W. H. Winsborough. Authenticated dictionaries for fresh attribute credentials. In *Trust Management: First International Conference, iTrust 2003 Heraklion, Crete, Greece, May 28–30, 2003 Proceedings 1*, 2003.

[70] M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*, volume 2. IEEE, 2001.

[71] M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Efficient authenticated data structures for graph connectivity and geometric search problems. *Algorithmica*, 60, 2011.

[72] Key transparency. https://github.com/google/keytransparency.

[73] V. Goyal. Reducing trust in the pkg in identity based cryptosystems. In *CRYPTO*, 2007.

[74] J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In M. Abe, editor, *Asiacrypt*, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[75] J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Asiacrypt*, 2010.

[76] J. Groth. On the size of pairing-based non-interactive arguments. In *Eurocrypt*, 2016.

[77] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais. Sok: Layer-two blockchain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 201–226. Springer, 2020.

[78] S. Haber and W. S. Stornetta. *How to time-stamp a digital document*. Springer, 1991.

[79] M. P. Heinl and V. Embacher. Bt2x: Multi-leveled binary transparency to protect the software supply chain of operational technology. In *Proceedings of the Sixth Workshop on CPS&IoT Security and Privacy*, pages 41–54, 2024.

[80] A. Hicks. Transparency, compliance, and contestability when code is law. *arXiv preprint arXiv:2205.03925*, 2022.

[81] B. Hof and G. Carle. Software distribution transparency and auditability. *arXiv preprint arXiv:1711.07278*, 2017.

[82] Y. Hu, K. Hooshmand, H. Kalidhindi, S. J. Yang, and R. A. Popa. Merklê 2: A low-latency transparency log system. In *IEEE Security & Privacy*. IEEE, 2021.

[83] W. Huang, J. Lin, Q. Wang, Y. Teng, H. Wan, and W. Wang. Certificate transparency for ecqv implicit certificates. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE, 2021.

[84] immudb. https://immudb.io/, 2025.

[85] A. Juels and B. S. K. Jr. Pors: proofs of retrievability for large files. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 584–597. ACM, 2007.

[86] Keybase chat. book.keybase.io/docs/chat.

[87] B. Kondracki, J. So, and N. Nikiforakis. Uninvited guests: Analyzing the identity and behavior of certificate transparency bots. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 53–70, 2022.

[88] N. Korzhitskii and N. Carlsson. Revocation statuses on the internet. In *Passive and Active Measurement: 22nd International Conference, PAM 2021, Virtual Event, March 29–April 1, 2021, Proceedings 22*, 2021.

[89] N. Korzhitskii, M. Nemec, and N. Carlsson. Postcertificates for revocation transparency. *arXiv preprint arXiv:2203.02280*, 2022.

[90] A. Kothapalli, S. Setty, and I. Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *CRYPTO*, 2022.

[91] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar. Certledger: A new pki model with certificate transparency based on blockchain. *Computers & Security*, 85, 2019.

[92] J. Kuszmaul. Verkle trees. Technical report, Massachusetts Institute of Technology, 2019.

[93] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *CACM*, 21(7), 1978.

[94] B. Laurie. Certificate transparency. *Communications of the ACM*, 57(10), 2014.

[95] B. Laurie and E. Kasper. Revocation transparency. *Google Research, September*, 33, 2012.

[96] H. Leibowitz, H. Ghalwash, E. Syta, and A. Herzberg. CTng: Secure Certificate and Revocation Transparency. Cryptology ePrint Archive, Paper 2021/818, 2021.

[97] J. Len, M. Chase, E. Ghosh, D. Jost, B. Kesavan, and A. Marcedone. Elektra: Efficient lightweight multi-device key transparency. In *ACM CCS*, 2023.

[98] J. Len, M. Chase, E. Ghosh, K. Laine, and R. C. Moreno. Optiks: An optimized key transparency system. Cryptology ePrint Archive, Paper 2023/1515, 2023.

[99] J. Len, E. Ghosh, P. Grubbs, and P. Rösler. Interoperability in End-to-End Encrypted Messaging. Cryptology ePrint Archive, Paper 2023/386, 2023.

[100] D. Leung, Y. Gilad, S. Gorbunov, L. Reyzin, and N. Zeldovich. Aardvark: An asynchronous authenticated dictionary with applications to account-based cryptocurrencies. In *USENIX Security*, 2022.

[101] K. Lewi and S. Lawlor. Introducing auditable key transparency for end-to-end encrypted messaging. https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/, Apr. 2023. Meta Engineering Blog.

[102] C. Li, S. M. Beillahi, G. Yang, M. Wu, W. Xu, and F. Long. Lvmt: An efficient authenticated storage for blockchain. *ACM Transactions on Storage*, 20(3):1–34, 2024.

[103] J. Li, M. N. Krohn, D. Mazieres, and D. E. Shasha. Secure Untrusted Data Repository (SUNDR). In *Osdi*, volume 4, 2004.

[104] H. Lipmaa. *Secure and efficient time-stamping systems*. Tartu University Press Tartu, 1999.

[105] K. T. W. G. M. List. Re: [keytrans] third-party auditing deployment timestamping vs validating. Key Transparency Working Group Mailing List, May 2024. https://mailarchive.ietf.org/arch/msg/keytrans/40IndRZ-37MhSP_TpIBJyqJhLzE/.

[106] H. Malvai, F. Falzon, A. Zitek-Estrada, and J. Bonneau. submission_crawler. https://github.com/Jasleen1/submission_crawler.git.

[107] H. Malvai, L. Kokoris-Kogias, A. Sonnino, E. Ghosh, E. Oztürk, K. Lewi, and S. Lawlor. Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging. *NDSS*, 2023.

[108] P. Maniatis and M. Baker. Secure history preservation through timeline entanglement. In *USENIX Security*, 2002.

[109] C. U. Martel, G. Nuckolls, P. T. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.

[110] G. Maxwell. Proving your bitcoin reserves. https://bitcointalk.org/index.php?topic=595180.0. Accessed: 2025-08-05.

[111] S. Meiklejohn, J. DeBlasio, D. O'Brien, C. Thompson, K. Yeo, and E. Stark. Sok: Sct auditing in certificate transparency. *arXiv preprint arXiv:2203.01661*, 2022.

[112] S. Meiklejohn, P. Kalinnikov, C. S. Lin, M. Hutchinson, G. Belvin, M. Raykova, and A. Cutter. Think global, act local: Gossip and client audits in verifiable data structures. *arXiv preprint arXiv:2011.04551*, 2020.

[113] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman. CONIKS: Bringing Key Transparency to End Users. In *USENIX Security*, 2015.

[114] T. Meunier and M. Galicer. Cloudflare helps verify the security of end-to-end encrypted messages by auditing key transparency for WhatsApp. https://blog.cloudflare.com/key-transparency/, 2024.

[115] A. Meyer. Better prefix authentication, 2023.

[116] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *FOCS*, 1999.

[117] A. Miller, M. Hicks, J. Katz, and E. Shi. Authenticated data structures, generically. *ACM SIGPLAN Notices*, 49(1), 2014.

[118] T. Mueller. Let's re-sign! analysis and equivocation-resistant distribution of openpgp revocations. In *ICOIN*. IEEE, 2022.

[119] T. Mueller, M. Stübs, and H. Federrath. Let's revoke! mitigating revocation equivocation by re-purposing the certificate transparency log. *Open Identity Summit 2019*, 2019.

[120] L. Nguyen. Accumulators from bilinear pairings and applications. In *CT˙RSA*, 2005.

[121] K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, I. Khoffi, J. Cappos, and B. Ford. Chainiac: Proactive software-update transparency via collectively signed skipchains and verified builds. In *USENIX Security*, 2017.

[122] K. Nissim and M. Naor. Certificate revocation and certificate update. In A. D. Rubin, editor, *Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, USA, January 26-29, 1998*. USENIX Association, 1998.

[123] A. Nitulescu. Sok: Vector commitments. *URL: https://www. di. ens. fr/˜ nitulesc/files/vc-sok. pdf*, 2021.

[124] C. of Digital Commerce. Proof of reserves – establishing best practices to build trust in the digital assets industry. https://digitalchamber.org /proof-of-reserves-blog/.

[125] A. Oprea and K. D. Bowers. Authentic time-stamps for archival storage. In *European Symposium on Research in Computer Security*, pages 136–151. Springer, 2009.

[126] A. Oprea and M. K. Reiter. Integrity checking in cryptographic file systems with constant trusted storage. In *USENIX Security Symposium*, pages 183–198. Boston, MA;, 2007.

[127] M. Paik, J. Irazábal, D. Zimmer, M. Meloni, and V. Padurean. immudb: A lightweight, performant immutable database. https://codenotary.s 3.amazonaws.com/Research-Paper-immudb-CodeNotary_v3.0.pdf, 2025.

[128] C. Papamanthou, E. Shi, R. Tamassia, and K. Yi. Streaming authenticated data structures. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2013.

[129] C. Papamanthou and R. Tamassia. Time and space efficient algorithms for two-party authenticated data structures. In S. Qing, H. Imai, and G. Wang, editors, *Information and Communications Security, 9th International Conference, ICICS 2007, Zhengzhou, China, December 12-15, 2007, Proceedings*, volume 4861 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007.

[130] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Authenticated hash tables. In *ACM CCS*, 2008.

[131] R. Peeters and T. Pulls. Insynd: Improved privacy-preserving transparency logging. In I. G. Askoxylakis, S. Ioannidis, S. K. Katsikas, and C. Meadows, editors, *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II*, volume 9879 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2016.

[132] Y. Peng, Z. Fan, B. Choi, J. Xu, and S. S. Bhowmick. Authenticated subgraph similarity search in outsourced graph databases. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1838–1860, 2015.

[133] T. Pulls and R. Peeters. Balloon: A forward-secure append-only persistent authenticated data structure. In G. Pernul, P. Y. A. Ryan, and E. R. Weippl, editors, *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*, volume 9327 of *Lecture Notes in Computer Science*, pages 622–641. Springer, 2015.

[134] R. Richmond. An Attack Sheds Light on Internet Security Holes. *The New York Times*, April 2011.

[135] T. Robert, A. Florin, A. David, and S. Emil. The challenges of proving solvency while preserving privacy. Cryptology ePrint Archive, Paper 2023/079, 2023.

[136] M. Rosenberg, T. Mopuri, H. Hafezi, I. Miers, and P. Mishra. Hekaton: Horizontally-scalable zkSNARKs via proof aggregation. Cryptology ePrint Archive, Paper 2024/1208, 2024.

[137] M. D. Ryan. Enhanced certificate transparency and end-to-end encrypted mail. *NDSS*, 2014.

[138] R. Sinha and M. Christodorescu. VeritasDB: High throughput key-value store with integrity. *Cryptology ePrint Archive*, 2025.

[139] C. Soghoian and S. Stamm. Certified Lies: Detecting and Defeating Government Interception Attacks against SSL (Short Paper). In *Financial Crypto*, 2012.

[140] L. Sorokin and U. Schoepp. Monitoring auditable claims in the cloud. *arXiv preprint arXiv:2312.12057*, 2023.

[141] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *IEEE Security & Privacy*. Ieee, 2016.

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

[143] A. Tomescu, I. Abraham, V. Buterin, J. Drake, D. Feist, and D. Khovratovich. Aggregatable subvector commitments for stateless cryptocurrencies. In *SCN*, 2020.

[144] A. Tomescu, V. Bhupatiraju, D. Papadopoulos, C. Papamanthou, N. Triandopoulos, and S. Devadas. Transparency logs via append-only authenticated dictionaries. In *ACM CCS*, 2019.

[145] A. Tomescu and S. Devadas. Catena: Efficient non-equivocation via bitcoin. In *IEEE Security & Privacy*. IEEE, 2017.

[146] A. Tomescu, Y. Xia, and Z. Newman. Authenticated Dictionaries with Cross-Incremental Proof (Dis)aggregation. Cryptology ePrint Archive, Paper 2020/1239, 2020.

[147] N. Tyagi, B. Fisch, A. Zitek, J. Bonneau, and S. Tessaro. Versa: Verifiable registries with efficient client audits from rsa authenticated dictionaries. In *ACM CCS*, 2022.

[148] I. Tzialla, A. Kothapalli, B. Parno, and S. Setty. Transparency dictionaries with succinct proofs of correct operation. Cryptology ePrint Archive, Paper 2021/1263, 2021.

[149] P. Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC*, 2008.

[150] Z. Wang, J. Lin, Q. Cai, Q. Wang, D. Zha, and J. Jing. Blockchain-based certificate transparency and revocation transparency. *IEEE Transactions on Dependable and Secure Computing*, 19(1), 2020.

[151] G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. ethereum.github.io/yellowpaper/paper.pdf, 2014.

[152] Q. Xing, X. Wang, X. Xu, J. Lin, F. Wang, C. Li, and B. Wang. Brt: An efficient and scalable blockchain-based revocation transparency system for tls connections. *Sensors*, 23(21):8816, 2023.

[153] X. Yang, Y. Zhang, S. Wang, B. Yu, F. Li, Y. Li, and W. Yan. Ledgerdb: A centralized ledger database for universal audit and verification. *Proceedings of the VLDB Endowment*, 13(12):3138–3151, 2020.

[154] M. L. Yiu, Y. Lin, and K. Mouratidis. Efficient verification of shortest path search via authenticated hints. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 237–248, 2010.

[155] J. Yu, M. Ryan, and C. Cremers. Decim: Detecting endpoint compromise in messaging. *IEEE Transactions on Information Forensics and Security*, 13(1), 2017.

[156] C. Yue, T. T. A. Dinh, Z. Xie, M. Zhang, G. Chen, B. C. Ooi, and X. Xiao. Glassdb: An efficient verifiable ledger database system through transparency. *arXiv preprint arXiv:2207.00944*, 2022.

[157] C. Yue, M. Zhang, C. Zhu, G. Chen, D. Loghin, and B. C. Ooi. Veribench: Analyzing the performance of database systems with verifiability. *Proceedings of the VLDB Endowment*, 16(9):2145–2157, 2023.

## APPENDIX

### A. Formal security definitions

**Definition 3.** *A historical stateful AD* AuthDS *is said to be history binding if the following probability is negligible in the security parameter for any PPT adversary* $\mathcal{A}$

$$
\begin{aligned}
Pr[\mathsf{pp} \xleftarrow{\$} \mathsf{AuthDS.Init}(1^\lambda),\\
(t, n, \mathsf{com}_t, (\mathsf{com}_{t+i}, \pi_i)_{i=1}^n,\\
\mathsf{key}, \mathsf{value}, \{(t_i, \mathsf{value}_i)\}_{i=1}^k, \phi_1, \phi_2, k, m) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}) :\\
(\mathsf{AuthDS.VerifyUpd}(\mathsf{pp}, \mathsf{com}_t,\\
(\mathsf{com}_{t+i}, \pi_i)_{i=1}^n, t, n) = 1)\\
\wedge m \in [1, k] \wedge \forall\, i \in [1, k],\, t < t_i \leq (t + n)\\
\wedge\, t_i + 1 = t_{i+1} \wedge (\mathsf{value} \neq \mathsf{value}_m)\\
\wedge \mathsf{AuthDS.LookupVerify}(\mathsf{pp}, \mathsf{com}_{t_m}, \phi_1, \mathsf{key}, \mathsf{value}) = 1\\
\wedge \mathsf{AuthDS.VerifyHistory}(\mathsf{pp}, \phi_2, (\mathsf{com}_{t_i})_{i=1}^k,\\
\{(t_i, \mathsf{value}_i)\}_{i=1}^k) = 1\}]
\end{aligned}
$$

**Definition 4.** *A stateful AD,* AuthDS *is said to be value binding if the following probability is negligible in the security parameter for any PPT adversary* $\mathcal{A}$

$$\begin{aligned}
&Pr[\mathsf{pp} \xleftarrow{\$} \mathsf{AuthDS.Init}(1^\lambda),\\
&(t, n, \mathsf{com}_t, (\mathsf{com}_{t+i}, \pi_i)_{i=1}^n,\\
&\mathsf{key}, \mathsf{value}_1, \mathsf{value}_2, \phi_1, \phi_2, j) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}) :\\
&(\mathsf{AuthDS.VerifyUpd}(\mathsf{pp}, \mathsf{com}_t,\\
&(\mathsf{com}_{t+i}, \pi_i)_{i=1}^n, t, n) = 1) \wedge (\mathsf{value}_1 \neq \mathsf{value}_2)\\
&\wedge t < j \leq t + n\\
&\wedge \mathsf{AuthDS.LookupVerify}(\mathsf{pp}, \mathsf{com}_j, \phi_1, \mathsf{key}, \mathsf{value}_1) = 1\\
&\wedge \mathsf{AuthDS.LookupVerify}(\mathsf{pp}, \mathsf{com}_j, \phi_2, \mathsf{key}, \mathsf{value}_2) = 1)\}]
\end{aligned}$$

**Definition 5.** *A stateful AD,* AuthDS *is said to have read-write consistency if, for any PPT adversary* $\mathcal{A}$

$$\begin{aligned}
&Pr[\mathsf{pp} \xleftarrow{\$} \mathsf{AuthDS.Init}(1^\lambda),\\
&(t, n, \mathsf{com}_t, (\mathsf{com}_{t+i}, \pi_i)_{i=1}^n, \mathsf{key}, j, k\\
&(\mathsf{op}_1, \phi_1^{\mathsf{Update}}), (\mathsf{op}_2, \phi_2^{\mathsf{Update}}),\\
&\mathsf{value}_1, \mathsf{value}_2, \phi^{\mathsf{Lookup}}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}) :\\
&(\mathsf{AuthDS.VerifyUpd}(\mathsf{pp}, \mathsf{com}_t,\\
&(\mathsf{com}_{t+i}, \pi_i)_{i=1}^n, t, n) = 1)\\
&\wedge \mathsf{AuthDS.MonitorUpdate}(\mathsf{pp}, \mathsf{com}_t, \mathsf{com}_{t+1},\\
&(\mathsf{op}_1, \mathsf{key}, \mathsf{value}_1), \phi_1^{\mathsf{Update}}) = 1\\
&\wedge t < j \leq t + n\\
&\wedge \mathsf{AuthDS.LookupVerify}(\mathsf{pp}, \mathsf{com}_j, \phi^{\mathsf{Lookup}},\\
&\mathsf{key}, \mathsf{value}_2) = 1\\
&\wedge \neg(t < k < j\\
&\wedge \mathsf{AuthDS.MonitorUpdate}(\mathsf{pp}, \mathsf{com}_k, \mathsf{com}_{k+1},\\
&(\mathsf{op}_2, \mathsf{key}, \mathsf{value}_2), \phi_2^{\mathsf{Update}}) = 1)\\
&\wedge \neg(\mathsf{value}_1 = \mathsf{value}_2)\\
&] \leq \mathsf{negl}(\lambda)
\end{aligned}$$

### *B. Further details on paper compilation*

Here we give further details on how we gathered the literature to survey using our script.

**Details of paper search.** Our paper search script utilizes Semantic Scholar's citations API to compile a list of the papers that cited the papers at our selected venues. As of this writing, unfortunately, Semantic Scholar's references API is not working, so we were not able to also compile the backward references of our published papers. Our script then selects the papers from our compilation which include, in their title or abstract, a set of keywords frequently used to denote authenticated dictionaries. Our script also deduplicates papers where possible, using the Semantic Scholar paper ID. This simple heuristic seems quite effective as we only found 6 duplicates in a list of 98 papers, and that too for papers which changed their titles at some point after being made public.

**Keywords.** For creating our list of keywords, we combined adjectives often used synonymously with "authenticated" with nouns used commonly to denote a key-value store. We also included a few applications that commonly use authenticated dictionaries in the literature. As our keywords, we included the following list, together with their plural version, where applicable: verifiable directory, verifiable dictionary, verifiable key-value store, verifiable key directory, authenticated key-value store, authenticated dictionary, authenticated directory, key-value database with integrity, database with integrity, dictionary with integrity, transparency log, key/binary/revocation/certificate transparency.

**Venues.** The venues we selected include top security conferences which tend to include applied cryptography papers, two databases conferences and two self-publishing venues. Our list was as follows: Network and Distributed System Security Symposium (NDSS), Conference on Computer and Communications Security (CCS), IEEE Symposium on Security and Privacy (Oakland), USENIX Security Symposium, Proceedings on Privacy Enhancing Technologies (PETS), Proceedings of the VLDB Endowment (VLDB), ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (SIGMOD), IACR Cryptology ePrint Archive, arXiv.org.

### *C. Additional cryptographic assumptions*

Different constructions of ADs rely on varying cryptogrpahic assumptions, as listed in Table II. All constructions rely on collision-resistant hash functions (CRHFs), so we omit this from Table II. The works we categorize as algebraic (or at least partly algebraic) constructions [144], [147], [148], [92], rely on additional assumptions, specifically:

- Verkle trees [92] rely on an assumption called the $q-$ Strong Billinear Diffie Hellman (qSBDH) assumption [73].
- AAD [144] also relies on the qSBDH assumption but also relies on an assumption know as the $q-$power knowledge of exponent (qPKE) assumption [75].
- UAD [146] and VeRSA [147] both rely on the strong RSA (sRSA) assumption.
- VeRSA and Verdict [148] both also use customized succinct non-interactive arguments of knowledge (SNARKs) [14] to improve the time complexity of Audit operations, and hence inherit their cryptographic assumptions. VeRSA uses Groth'16 [76], which requires the qSBDH assumption. Verdict builds on Nova [90], adding the related, Strong External Diffie Hellman (SXDH) [32] assumption.

As we discuss in Sec. V-C, several works [113], [28], [107], [98], [97], [31] rely on algebraic assumptions for the security of verifiable random functions (VRFs), which they only rely on for privacy (but not history binding). Hence we do not categorize them as algebraic in general, as these assumptions are not required in applications which don't require privacy. Where VRFs are needed, they can be built from various cryptographic assumptions, including the decisional Diffie-Hellman (DDH) assumption [65], the computational Diffie-Hellman assumption in a gap group [19], or RSA assumptions [116]. There is also a construction of VRFs from any unique signature scheme, in the random oracle model [116].