SoK: Authenticated Prefix Relations —
A Unified Perspective On Relative Time-Stamping and Append-Only Logs

Aljoscha Meyer
Technical University Berlin

Abstract

Secure relative timestamping and secure append-only logs are two historically mostly independent lines of research, which we show to be sides of the same coin — the authentication of prefix relations. From this more general viewpoint, we derive several complexity criteria not yet considered in previous literature. We define transitive prefix authentication graphs, a graph class that captures all hash-based timestamping and log designs we know of. We survey existing schemes by expressing them as transitive prefix authentication graphs, which yields more compact definitions and more complete evaluations than in the existing literature.

1 Introduction

Consider the problem of mapping any finite sequence to a small digest, such that for any pair of a sequence and one of its prefixes, their digests together with a small certificate unforgeably certify that one is a prefix of the other. In other words, consider the problem of finding an authenticated data structure for the prefix relation on strings.

While we are not aware of any prior work that explicitly takes this abstract viewpoint of the problem, there are several publications that tackle it through the lens of a specific use-case: secure logging [39] [11] [37], accountable shared storage [25] [49], certificate transparency [20] [21] [23], or data replication [33] [43].

Some fifteen years before most of these publications, investigation into secure relative timestamping – given two events, give an unforgeable certificate that one happened before the other – has yielded *linking schemes* [7] as a class of efficient timestamping solutions, with several specific schemes being proposed [17] [5] [8]. While solutions to relative timestamping need not necessarily yield solutions to prefix authentication in general, every *linking scheme* does provide prefix authentication.

So there are quite a few disjoint publications working on essentially the same problem, but with sparse interreferencing and the independent invention of various metaphorically wheel-shaped devices. All these techniques are based on the observation that if some object contains a secure hash of another object, the prior must have been created after the latter. Linking schemes are a well-defined class of such solutions, other approaches only have ad-hoc descriptions and proofs of correctness. Different publications also considered different efficiency criteria, making it difficult to objectively compare several approaches.

We provide systemization on several levels. First, we give a precise definition of prefix authentication, and systematically derive a set of efficiency criteria from this definition (section 4). Second, we define a class of digraphs which generalizes the linking schemes, can be used for prefix authentication, and includes all hash-based prefix authentication schemes that we are aware of (section 5). And third, we survey those prior schemes, expressing them in our formalism and evaluating them by our complexity criteria, yielding the most complete comparison of such schemes so far (section 6), and revealing some flaws in the current state of the art.

We round out our presentation with an overview of related work (section 2), some mathematical preliminaries (section 3), and a conclusion (section 7).

2 Related Work

Authenticated data structures [42] are data structures which allow to supplement query results with a small *certificate* of the result's validity; a *verifier* with access to only a small *digest* of the current state of the data structure can check whether the query was answered truthfully based on the certificate.

Secure (relative) timestamping asks for authenticated data structures for a happened-before relation; given two events, it should be possible to prove which occurred first (as opposed to both happening concurrently). Haber and Stornetta's foundational work [17] arranges events in a linked list, using secure hashes as references. The path from an event to a prior one certifies their happened-before relation.

A straightforward optimization is batching multiple events into a single *round* by storing all events within a round in

a Merkle tree, and linearly linking the roots of each round rather than individual events [3]. This is fully analogous to the blocks of a blockchain. Certificate sizes do not decrease asymptotically however.

Buldas, Laud, Lipmaa, and Villemson provide the first asymptotic improvement by adding additional hashes such that there exist paths of logarithmic length between any pair of events [7]. After Buldas and Laud finding the solution with the shortest certificates in this class [5], Buldas, Lipmaa, and Schoenmakers provide *threaded authentication trees* [8], an even more efficient construction in the class of acyclic graphs where the vertex for every event is reachable from all later events. This is the class we refer to as *linking schemes*, and upon which our generalization builds.

Their presentation and optimality proofs rely on a notion of time-stamping rounds, the maximum number of events in a single round features in their complexity analyses. We take on a more general setting, the notion of rounds corresponds to prefix authentication for sequences of bound length. Hence, their lower bounds do not apply to our setting.

Laurie, Langley, Kasper, Messeri, and Stradling introduce *certificate transparency* (CT) [20] [21] [23], a proposed Internet standard for publicly logging information about the activities of *certificate authorities* (CAs). Rather than detailing the extensive background of *public-key infrastructure* into which CT embeds itself, we refer to [24] Section 2. We shall further abstract over optimization details of CT such as *signed certificate timestamps* (SCTs) or *maximum merge delays*. What remains is a so-called[1] append-only log with the dual-purpose of certifying both a prefix relation and set membership. The CT data structure achieves the same certificate sizes as the threaded authentication trees, but is not a linking scheme, which prompts our generalization.

Various publications propose alternatives to the original CT design, reasons include adding support for certificate revocation, strengthening the trust model, or mitigating privacy concerns [22] [38] [18] [27] [2] [48] [24]. Other publications extend the concept of logging objects for accountability reasons to application binaries [15] [31] [1] or arbitrary data [14] [36]. None of them improve the underlying prefix authentication scheme.

Several publications generalize (and formalize) the requirements behind CT as the combination of authenticating an append-only property and supporting authenticated membership queries: the *logging scheme* [13], the *dynamic list com-*

mitment [9], or the *append-only authenticated dictionary* [44] fall in this category. We are effectively "factoring out" the prefix authentication; the other "factor" being authenticated set data structures, which have already received extensive treatment [30] [16] [6] [34] on their own. Usually so, our definition requires neither a specific *append* operation nor "append-only proofs"; the general notion of prefixes suffices.

Secure scuttlebutt [43] and *hypercore* [33] rely on authenticated prefix relations for efficient event replication. Secure scuttlebutt uses a linked list, hypercore has its own solution which is almost identical to the log of CT.

Some approaches to tamper-evident logging also rely on linked lists [39] [37]. In this context, Crosby and Wallach [11] designed the first non-linking scheme approach to prefix authentication that we know of. Their scheme precedes the CT design by five years, but it is strictly more complex and inefficient.

Work on secure networked memory has also made use of linked lists for prefix authentication [25] [49]. They speak of *fork-consistency*: because forks (creation of strings neither of which is a prefix of the other) cannot be authenticated, a malicious author must consistently feed updates from the same forks to the same data consumers to avoid detection.

Since a malicious data source can easily present such different views to several consumers, data consumers should exchange information amongst each other to protect against such *split world* attacks. Epidemic protocols [12] can be used to this end [10] [32]. A complementary approach is to enforce (efficient) *cosigning*, where a data source must present its updates to a large number of other participants for approval [41].

Various authors argue that such reactive detection mechanism are insufficient, and propose a proactive approach based on enforcing global consensus by moving events onto a blockchain [45] [31] [1] [26] [19] [46]. Why an adversary with enough power to perpetually prevent communication between nodes that received incomparable views would be unable keep those nodes in distinct bubbles that each produce separate extensions of the blockchain is beyond us, but who are we to argue against the magic powers of the blockchain?

While we restrict our attention to prefix authentication schemes that exclusively rely on secure hashing, there also exist approaches based on cryptographic accumulators [40].

3 Preliminaries

We write $\mathbb{N}_0$ for the natural numbers, and $\mathbb{N}$ for the natural numbers without 0. Let $N \subseteq \mathbb{N}_0$, and $n \in \mathbb{N}_0$, then $N^{\leq n} := \{m \in N : m \leq n\}$ and $N^{\geq n} := \{m \in N : m \geq n\}$. We denote by $\mathsf{bits}(n)$ the unique set of natural numbers such that $\sum_{k \in \mathsf{bits}(n)} 2^k = n$.

We assume basic understanding of cryptographic hash functions [28], and a basic background in graph theory [47]. In the following, u, v always denote vertices, U, X always denote sets of vertices, G always denotes a graph on vertices V with

---

[1] These logs are, quite simply, *not* append-*only* logs. A log maintainer cannot be prevented from creating several branches, this can merely be detected after the fact. Upon detection, a log consumer must somehow deal with the misbehavior, for example, by invalidating the log. But at that point, the log has turned into an "append-only-until-a-full-deletion" log, which is strictly more powerful than an append-only log. No matter how forks are handled, the data structure is too powerful. Calling a data structure an append-*only* log even though it is *not* is careless at best and misleading at worst, especially in a security context. Hence, we exclusively talk about prefix authentication from this point on.

edges E. As we talk about directed graphs exclusively, all graph terminology (*graph*, *path*, etc.) refers to directed concepts. We only consider graphs without loops. Whenever we apply a concept that is defined on sets of vertices to an individual vertex, we mean the concept applied to the singleton set containing that vertex.

A *directed acyclic graph* (DAG) is a graph without cycles. The (open) *out-neighborhood* of U in a graph G is $N_G^+(U) := \{v \in V \setminus U : \text{there is } u \in U \text{ such that } (u,v) \in E\}$. A *sink* is a vertex with an empty out-neighborhood, $\mathsf{sinks}\,G$ denotes the sinks of G. $\mathsf{reach}_G(v)$ denotes the set of all vertices $u \in \mathsf{V}(G)$ such that there is a path from v to u in G, and $\mathsf{reach}_G(U) := \bigcup_{u \in U} \mathsf{reach}_G(u)$.

We use *sequence* and *string* synonymously and assume both to always be finite. "·" denotes concatenation, $\preceq$ denotes the prefix relation, for $s \preceq t$, we call s a *prefix* of t and t an extension of s. ε denotes the empty sequences. For a sequence s, s_i denotes the i-th sequence item, with indexing startig at 1.

4 Prefix Authentication Schemes

We can now state what it means to authenticate a prefix relation.

Definition 1 (Prefix Authentication Scheme). A *prefix authentication scheme* (*PAS*) for sequences over some universe U is a triplet of algorithms digest, $\mathsf{certify}$, and verify:

- $\mathsf{digest} : U^* \to \{0,1\}^*$ maps any sequence s to some bitstring $\mathsf{digest}(s)$, the *digest* of s.

- $\mathsf{certify} : U^* \times U^* \rightharpoonup \{0,1\}^*$ maps any pair of sequences $s \preceq t$ to some bitstring $\mathsf{certify}(s,t)$, the *prefix certificate* of s and t.

- verify takes bitstrings d_s, d_s and p and two natural numbers len_s and len_t, and returns $\mathtt{true}$ if there exist sequences $s \preceq t$ of length len_s and len_t respectively such that $d_s = \mathsf{digest}(s)$, $d_s = \mathsf{digest}(t)$, and $p = \mathsf{certify}(s,t)$. Furthermore, it must be computationally infeasible to find inputs for verify that result in $\mathtt{true}$ otherwise.

From this definition, we can systematically derive the efficiency criteria by which to judge a PAS. Straightforward criteria are the time and space complexity of the three functions, as well as the sizes of digests and prefix certificates. Previous work often ignores some of these, especially when they are obvious in the context of that work. In comparing several different prefix authentication approaches from fully independent work, we consider it important to make all these basic criteria explicit.

A less obvious pair of criteria derives from the question of which portions of their input the algorithms actually utilize. Both digest and $\mathsf{certify}$ receive full sequences as input. In practice however, we are interested in schemes that compute

them from only a sublinear amount of information about the sequence, say, in a peer-to-peer system where storing full sequences would impose prohibitive storage overhead.

The first such criterium asks which information is required to indefinitely keep appending items to a sequence and compute the digests of all these extensions. We require a function $\mathsf{iterative\_digest} : I \times U \to \{0,1\}^* \times I$ that maps information (of some type I) about a sequence s and a new item u to the digest of $s \cdot u$ and the information (again of type I) about $s \cdot u$. Repeatedly calling this function allows computing the digest for any sequence. Besides the computational complexity of $\mathsf{iterative\_digest}$, we are interested in the size of the information for sequences of length n.

For prefix certificate computation, we wish to map any sequence to some (small) piece of information such that the prefix certificate for any two sequences can be computed from their two pieces of information. Again we are interested in the cost of these computations, and the size of the information depending on the length of the sequence. This generalizes the notion of a *time certificate* [8], which is the primary focus of the timestamping literature, whereas this criterium is rarely analyzed in the discussion of logs. We call the piece of information about each sequence its *positional certificate*.

5 A Class of Solutions

We now develop a family of prefix authentication schemes that generalizes over all hash-based secure timestamping and logging schemes that we are aware of.

When some object contains a secure hash of another object, any change to the latter would invalidate the prior. A classic data structure to leverage this property is the *Merkle tree* [29], a rooted tree which labels leaves with a secure hash of their contained value, and which labels inner vertices with a secure hash of the concatenation of the child labels.

We generalize the idea behind Merkle trees to arbitrary DAGs. We label sinks via some function $\mathsf{l} : \mathsf{sinks}(V) \to \{0,1\}^k$. For all non-sink vertices, we aggregate the labels of their out-neighbors via some hash function h. In order to deterministically apply h to sets (of out-neighbors of a vertex), we assume there is some arbitrary but fixed total order $\leq$ on V, and define how to convert any vertex set U into a unique sequence $\mathsf{seq}(U)$ via $\mathsf{seq}(\emptyset) := \varepsilon$, $\mathsf{seq}(U) := \min_{\leq}(U) \cdot (U \setminus \min_{\leq}(U))$. We then define

$$\mathsf{label}_{\mathsf{l,h}}(v) := \begin{cases} \mathsf{l}(v) & \text{if } v \in \mathsf{sinks}(V), \\ \mathsf{h}(\mathsf{seq}(N_G^+(v))) & \text{otherwise.} \end{cases}$$

For binary out-trees, this yields exactly the Merkle tree construction. We call a pair $(G = (V,E), \mathsf{label}_{\mathsf{l,h}})$ a *Merkle DAG*. If every maximal path from v intersects U, then $\mathsf{label}_{\mathsf{l,h}}(v)$ can be computed from the labels of the vertices in U; we say that U *label-determines* v. If U label-determines every $x \in X$,

we say that U label-determines X. Given U and v such that U label-determines v, and some U-labeling $p : U \to \{0,1\}^k$, we denote the expected label of v that can be computed from U and p by $\mathsf{label}_\mathsf{h}^p(v)$. Observe that functions p are practically unique for any fixed $\mathsf{label}_\mathsf{h}^p(v)$:

Proposition 1. Let $(G = (V,E), \mathsf{label}_{\mathsf{l},\mathsf{h}})$ be a Merkle DAG, let $v \in V$, and $U \subseteq V$ such that U label-determines v. Then it is computationally infeasible to find a labeling $p : U \to \{0,1\}^k$ such that $\mathsf{label}_\mathsf{h}^p(v) = \mathsf{label}_{\mathsf{l},\mathsf{h}}(v)$ and $p \neq \mathsf{label}_{\mathsf{l},\mathsf{h}}|_{\mathsf{dom}(p)}$.

Proof. Assume it was feasible to find $p \neq \mathsf{label}_{\mathsf{l},\mathsf{h}}|_{\mathsf{dom}(p)}$ with $\mathsf{label}_\mathsf{h}^p(v) = \mathsf{label}_{\mathsf{l},\mathsf{h}}(v)$. Then there must have existed a vertex w with $\mathsf{label}_\mathsf{h}^p(w) = \mathsf{label}_{\mathsf{l},\mathsf{h}}(w)$ having an out-neighbor x with $\mathsf{label}_\mathsf{h}^p(x) \neq \mathsf{label}_{\mathsf{l},\mathsf{h}}(x)$. Hence two distinct inputs to h yielded the same hash, contradicting the collision resistance of h. $\square$

Our prefix authentication schemes will use Merkle DAGs whose sinks we each label with a secure hash of a sequence item. We say a vertex set U is a *commitment* to a vertex set X if $X \subseteq \mathsf{reach}(U)$. Changing the label of any vertex in X changes the label of at least one vertex in U. The digests of our schemes will be the labels of singleton commitments to vertices labeled by hashes of sequence items.

We say a vertex set U is a *tight commitment* to a vertex set X if U is a commitment to X and X label-determines U.

Our prefix certificates generalize the set membership proofs of classic Merkle trees. Merkle trees offer compact set membership proofs by reconstructing the label of the root vertex from the labels of the out-neighborhood of a path to a leaf. The out-neighborhood of a union of such paths certifies membership of several leaves at once. We can generalize this to arbitrary Merkle DAGs (see fig. 1 for an example):

Definition 2 (Subgraph Proof). Let $(G = (V,E), \mathsf{label}_{\mathsf{l},\mathsf{h}})$ be a Merkle DAG, let $U \subseteq V$, and let $v \in V$ such that $U \subseteq \mathsf{reach}_G(v)$. Let P be a family of paths starting in v such that $U \subseteq N_G^+(P)$, and let $p : N_G^+(P) \to \{0,1\}^k$, where $\{0,1\}^k$ is the codomain of h.

We then call $(\mathsf{label}_{\mathsf{l},\mathsf{h}}(v), p)$ a *potential subgraph proof* of U for v.

Observe that $N_G^+(P)$ label-determines v. We say $(\mathsf{label}_\mathsf{h}(v), p)$ is a *(verified) subgraph proof* if $\mathsf{label}_\mathsf{h}^p(v) = \mathsf{label}_{\mathsf{l},\mathsf{h}}(v)$, and a *refuted subgraph proof* otherwise.

Proposition 2. Let $(G = (V,E), \mathsf{label}_{\mathsf{l},\mathsf{h}})$ be a Merkle DAG, and let $v \in V$. Then, by proposition 1, it is computationally infeasible to find U and p such that $(\mathsf{label}_{\mathsf{l},\mathsf{h}}(v), p)$ is a verified subgraph proof of U for v with $p \neq \mathsf{label}_{\mathsf{l},\mathsf{h}}|_{\mathsf{dom}(p)}$.

Corollary 1. Let $(G = (V,E), \mathsf{label}_{\mathsf{l},\mathsf{h}})$ be a Merkle DAG, let $v \in V$, let $U \subseteq V$, and let $(\mathsf{label}_{\mathsf{l},\mathsf{h}}(v), p)$ be a subgraph proof of U for v. If h is secure, then $N_G^+(\mathsf{dom}(p))$ is a subgraph of G. In particular, $G[U]$ is a subgraph of G, and $U \subseteq \mathsf{reach}_G(v)$.
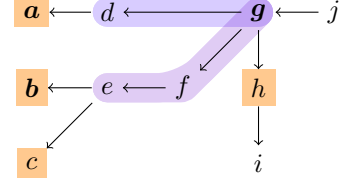

Figure 1: An example subgraph proof of $\{a, b\}$ for g. $P := \{d, e, f, g\}$ consists of the vertices of two paths starting in g whose out-neighborhoods together include $\{a, b\}$. The out-neighborhood of P and its labels yield p, and it (necessarily) label-determines g: the label of d can be computed from the label of a; the labels of b and c suffice to compute the label of e, which in turn determines the label of f; together with the label of h, we can compute the label of g. If this computed label matches the label given with the subgraph proof, we can be certain (up to hash collisions) that a and b relate to g in the graph as expected and indeed have the labels claimed by p.

5.1 Linking Schemes

We now have the terminology to define the *linking schemes*, a class of prefix authentication schemes based on Merkle DAGs. We use secure hashes of sequence items to label the sinks of some Merkle DAG in which the set of the sinks that correspond to any prefix of the sequence has a common ancestor vertex; the labels of the common ancestors serve as digests, and subgraph proofs between the digest vertices serve as prefix certificates.

First, we formalize the notion of mapping sequence items to sinks:

Definition 3 (Sequence Graph). Let G be an acyclic graph with $\mathsf{sinks}(G) \supseteq \mathbb{N}$, and let s be a sequence of length len_s.

The *sequence graph* of s and G is the Merkle DAG $(G, \mathsf{label}_{\mathsf{l}_s,\mathsf{h}})$ with

$$\mathsf{l}_s(s_v) := \begin{cases} \mathsf{h}(v) & \text{if } v \in \mathbb{N}^{<len_s}, \\ \mathsf{h}(\varepsilon) & \text{otherwise.} \end{cases}$$

Next, we can describe the class of graphs that allows for prefix authentication:

Definition 4 (Linking Scheme Graph). A graph $G = (V,E)$ is a *linking scheme graph* if G is acyclic, $\mathsf{sinks}(G) \supseteq \mathbb{N}$, and there exist functions $\mathsf{digest\_vertex} : \mathbb{N} \to V$, and $\mathsf{certificate\_vertices} : \mathbb{N} \times \mathbb{N} \to \mathcal{P}(V)$ such that for all $len_s, len_t \in \mathbb{N}$ with $len_s < len_t$:

- $\mathsf{digest\_vertex}(len_s)$ is a tight commitment to $\mathbb{N}^{\leq len_s}$, and

- $\mathsf{certificate\_vertices}(len_s, len_t)$ is a path starting in $\mathsf{digest\_vertex}(len_t)$ such that $\mathsf{digest\_vertex}(len_s) \in N_G^+(\mathsf{certificate\_vertices}(len_s, len_t))$.
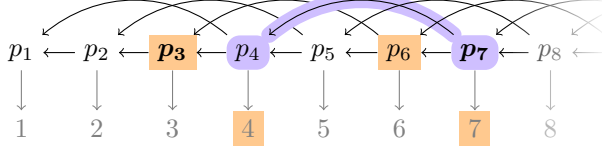
Figure 2: An example linking scheme, together with a certificate for $len_s := 3$ and $len_t := 7$. p_3 and p_7 are the digest vertices for 3 and 7 respectively. The path (p_7, p_4) is a (shortest) path from p_7 whose out-neighborhood contains p_3, and its out-neighborhood yields the prefix certificate.

Using a linking scheme graph as the underlying graph of a Merkle DAG yields a prefix authentication scheme (fig. 2 gives an example):

Definition 5 (Linking Scheme). Let $G = (V, E)$ be a linking scheme graph, and let h be a secure hash function.

G and h define a *linking scheme* (digest, commit, certify) using the following functions:

For strings s of length len_s, let $(G, \mathsf{label}_{\mathsf{ls,h}})$ be the sequence graph of s and G. We then define digest$(s) :=$ $\mathsf{label}_{\mathsf{ls,h}}(\mathsf{digest\_vertex}(len_s))$. Observe that this can be computed from s alone as $\mathbb{N}^{\leq len_s}$ label-determines digest_vertex(len_s).

For strings $s \preceq t$ of length len_s and len_t respectively, let $(G, \mathsf{label}_{\mathsf{lt,h}})$ be the sequence graph of t and G. We then define certify(s, t) as the bitstring obtained by sorting $N_G^+(\mathsf{certificate\_vertices}(len_s, len_t))$ according to $\leq$ and concatenating the labels (in $(G, \mathsf{label}_{\mathsf{lt,h}})$) of these vertices. Finally, we define verify$(d_s, d_t, p, len_s, len_t)$ to first parse p into a labeling p' of $N_G^+(\mathsf{certificate\_vertices}(len_s, len_t))$ and then return whether (d_t, p') is a verified subgraph proof of digest_vertex(len_s) for digest_vertex(len_t).

Proposition 3. Every linking scheme is a prefix authentication scheme.

Proof. All functions have the required signatures. Let $s \preceq t$ be sequences of length len_s and len_t respectively, then verify(digest(s), digest(t), certify(s,t), len_s, len_t) returns true by construction; and returning true for these inputs implies $s \preceq t$ by corollary 1. Any other inputs that yield true witness a hash collision by proposition 2. $\square$

This definition of linking schemes is adapted from Buldas et al. [8] and captures the linear linking scheme [17], antimonotone schemes [7] [5], and the threaded authentication trees [8]. This class of schemes does not however include Crosby and Wallach's secure logging scheme [11], transparency logs [20], or hypercore [33], prompting our search for a further generalization.

5.2 Transitive Prefix Authentication Schemes

The generalization to include these other schemes is simple[2]: rather than giving subgraph proofs that some digest vertex is reachable from another, we give subgraph proofs that some set that label-determines a digest vertex is reachable from another digest vertex.

Definition 6 (Transitive Prefix Authentication Graph). A graph $G = (V, E)$ is a *transitive prefix authentication graph* (TPAG) if G is acyclic, sinks$(G) \supseteq \mathbb{N}$, and there exist functions digest_vertex : $\mathbb{N} \to V$, commit : $\mathbb{N} \to \mathcal{P}(V)$, and certificate_vertices : $\mathbb{N} \times \mathbb{N} \to \mathcal{P}(V)$ such that for all $len_s, len_t \in \mathbb{N}$ with $len_s < len_t$:

- digest_vertex(len_s) is a tight commitment to $\mathbb{N}^{\leq len_s}$,

- commit(len_s) label-determines digest_vertex(len_s), and

- certificate_vertices(len_s, len_t) is a union of paths, each starting in digest_vertex(len_t), such that commit$(len_s) \subseteq N_G^+(\mathsf{certificate\_vertices}(len_s, len_t))$.

Using a TPAG as the underlying graph of a Merkle DAG yields a prefix authentication scheme (fig. 3 gives an example). Because the definition is similar to that of linking schemes, we have typeset the differences in a **bold** font.

Definition 7 (Transitive Prefix Authentication Scheme). Let $G = (V, E)$ be a **TPAG**, and let h be a secure hash function.

G and h define a *transitive prefix authentication scheme* (TPAS) (digest, commit, certify) using the following functions:

For strings s of length len_s, let $(G, \mathsf{label}_{\mathsf{ls,h}})$ be the sequence graph of s and G. We then define digest$(s) :=$ $\mathsf{label}_{\mathsf{ls,h}}(\mathsf{digest\_vertex}(len_s))$. Observe that this can be computed from s alone as $\mathbb{N}^{\leq len_s}$ label-determines digest_vertex(len_s).

For strings $s \preceq t$ of length len_s and len_t respectively, let $(G, \mathsf{label}_{\mathsf{lt,h}})$ be the sequence graph of t and G. We then define certify(s, t) as the bitstring obtained by sorting $N_G^+(\mathsf{certificate\_vertices}(len_s, len_t))$ according to $\leq$ and concatenating the labels (in $(G, \mathsf{label}_{\mathsf{lt,h}})$) of these vertices. Finally, we define verify$(d_s, d_t, p, len_s, len_t)$ to first parse p into a labeling p' of $N_G^+(\mathsf{certificate\_vertices}(len_s, len_t))$ and then return whether (d_t, p') is a verified subgraph proof of digest_vertex(len_s) for digest_vertex(len_t) **and whether** $\mathsf{label}_{\mathsf{h}}^{p'}(\mathsf{digest\_vertex}(len_s)) = d_s$.

Proposition 4. Every **TPAS** is a prefix authentication scheme.

Proof. All functions have the required signatures. Let $s \preceq t$ be sequences of length len_s and len_t respectively, then verify(digest(s), digest(t), certify(s,t), len_s, len_t)

---

[2]Simple, that is, when starting from a characterization of linking schemes that was specifically crafted to allow for this generalization.
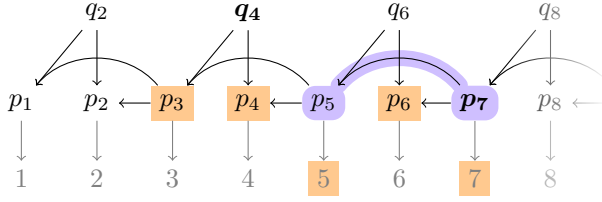
Figure 3: A TPAS, together with a certificate for $len_s := 4$ and $len_t := 7$. q_4 and p_7 are the digest vertices for 4 and 7 respectively. The path (p_7, p_5) is a (family of exactly one) path from p_7 whose out-neighborhood contains p_3 and p_4, who together label-determine q_4. Its out-neighborhood yields the prefix certificate.

returns `true` by construction; and returning `true` for these inputs implies $s \preceq t$ by corollary 1. Any other inputs that yield `true` witness a hash collision by proposition 2 (when verifying the subgraph proof) **or by proposition 1** (when checking that $\mathsf{label}_h^{p'}(\mathsf{digest\_vertex}(len_s)) = d_s$). □

5.3 Efficiency Criteria

Because every TPAS stems from a TPAG, we can reason about prefix authentication schemes while remaining solely in the realm of unlabeled digraphs; the labelings are a deterministic afterthought. Given the length k of each individual hash, we can also reason about the complexity criteria of section 4 by solely considering graph properties.

Since digests are labels of individual vertices, the digest size of any sequence is k.

Worst-case and average prefix certificate sizes correspond directly to the sizes of the out-neighborhood of $\mathsf{certificate\_vertices}(len_s, len_t)$: the prefix certificate is $\mathsf{label}_{l,h}|_{N_G^+(\mathsf{certificate\_vertices}(len_s, len_t))}$, which can be encoded by fixing an ordering on $\mathsf{V}(G)$ in advance and simply listing the labels of $N_G^+(\mathsf{certificate\_vertices}(len_s, len_t))$ according to that ordering. The space requirement is hence $|N_G^+(\mathsf{certificate\_vertices}(len_s, len_t))| * k$.

Computing verify consists of reconstructing $\mathsf{digest}(len_s)$ from $\mathsf{commit}(len_s)$ and $\mathsf{digest}(len_t)$ from $\mathsf{certify}(len_s, len_t)$. Reconstructing $\mathsf{digest}(len_s)$ from $\mathsf{commit}(len_s)$ requires time proportional to the number of distinct edges on all paths from $\mathsf{digest}(len_s)$ to any vertices from $\mathsf{commit}(len_s)$. Reconstructing $\mathsf{digest}(len_t)$ from $\mathsf{certify}(len_s, len_t)$ requires time proportional to the number of edges in the graph induced by the closed neighborhood of $\mathsf{certificate\_vertices}(len_s, len_t)$.

The complexity of computing $\mathsf{digest}(len_s)$ and $\mathsf{certify}(len_s, len_t)$ is less straightforward. Because $\mathsf{digest\_vertex}(len_s)$ is a common ancestor of all $i \leq len_s$, computing its label requires at least $O(len_s)$ time. In a

realistic setting, one would instead precompute and store the labels of all all digest vertices, turning the computation into a simple look-up. The additional storage cost of $O(len_s)$ space does not exceed the cost for storing len_s sequence items in the first place. But when also precomputing the labels for all vertices that can appear in any prefix certificate, the overall space complexity might exceed $O(len_s)$.

We simplify the analysis by looking at the storage cost for precomputing the labels of *all* vertices. Intuitively, we expect efficient schemes to not have redundant vertices, so this simplification should only be an overapproximation for schemes that are of little practical interest to begin with.

In order to classify the storage cost per sequence item, we define the graph $G[len_s] := \bigcup_{i \leq len_s} \mathsf{reach}(\mathsf{digest\_vertex}(i))$ of all vertices that are required to work with a sequence of length len_s. The number of labels that need to be precomputed and stored because of the len_s-th sequence item is then given by $|\mathsf{V}(G[len_s]) \setminus \mathsf{V}(G[len_s - 1])|$. We are both interested in the worst-case for any len_s and in the amortized case of averaging over the first len_s sequence items.

For iterative computation of digests, we require a function $\mathsf{digest\_pool} : \mathbb{N} \to \mathsf{V}(G)$ such that $\mathsf{digest\_pool}(len_s - 1) \cup \{len_s\}$ label-determines both $\mathsf{digest}(len_s)$ and all vertices in $\mathsf{digest\_pool}(len_s)$. Intuitively, the digest pool for some len_s consists of all the vertices in $G[len_s]$ whose label impacts the label of any future vertex, i.e., every vertice in $N^+(G[len_t] - G[len_s]) \cap G[len_s]$ for any $len_t > len_s$. The labels of $\mathsf{digest\_pool}(len_s)$ then allow for indefinitely appending new items to a sequence of length len_s by adding any newly created vertices whose labels will be used in the future; we are interested in functions that minimize $|\mathsf{digest\_pool}(len_s)|$.

For computing prefix certificates from only parts of full sequences, we consider functions $\mathsf{certificate\_pool} : \mathbb{N} \to \mathsf{V}(G)$ such that $N_G^+(\mathsf{certificate\_pool}(len_s)) \cup N_G^+(\mathsf{certificate\_pool}(len_t))$ label-determines $N_G^+(\mathsf{certificate\_vertices}(len_s, len_t))$. This allows for computing $\mathsf{certify}(len_s, len_t)$ from the (out-neighborhoods of the) *certificate pools* of len_s and len_t. We are then interested in functions that minimize $|N_G^+(\mathsf{certificate\_pool}(len_s))|$.

5.4 Secure Timestamping

Secure timestamping [17] asks to cryptographically certify the happened-before relation on some totally-ordered set of events. Prefix authentication does not immediately imply secure timestamping, but the problems are related: if event number s happened before event number t, we can certify that the sequence of the first s events is a prefix of the sequence of the first t events. This only relates the *digests* of the event sequences however, not the events *themselves*.

We can extend every TPAS to provide time stamping. We define the *identifier* of the $i - th$ item in some sequence as a

(deterministically selected) subgraph proof of its vertex for digest_vertex(i). We call it an *identifier* because it identifies a particular item as occuring at a particular position in a particular sequence (and its extensions).

Let s be the sequence of the first len_s events, and let t be the sequence of the first len_t events, with $s \preceq t$. To certify that event number len_s happened before event number len_t, simply provide certify(s,t) together with the identifiers of len_s and len_t. certify(s,t) certifies the happened-before relation of the digests, and the identifiers tie the digests to the actual items. Verification consists of verifying the certificate as well as the two identifiers.

Hence, the worst-case and average size of the identifier for any item at position n becomes another complexity parameter of interest. Its exact value is k (the size of an individual digest) times the number of vertices in the subgraph proof of $\{i\}$ for digest_vertex(i).

6 Prior Schemes

We now give definitions of timestamping and logging schemes from the literature, expressed as TPASs. This serves as a demonstration of the generality and applicability of TPASs, it provides a survey of existing approaches, and it allows us to apply our efficiency criteria to previous work.

Our definition of Merkle DAGs automatically incorporates the notion of *computing* the labels along a path from a prefix certificate rather than directly *using* those labels as the certificate, an optimization introduced by Buldas, Lipmaa, and Schoenmakers. [8] after several prior schemes had already been published. The improvement that their section 5.2 gives over the antimonotone linking schemes [7] [5] is inherent to our formulations of *all* approaches.

Our presentation of timestamping schemes differs significantly from their original presentation in that we do not consider a setting of timestamping rounds. Working with timestamping rounds effectively amounts to solving prefix authentication for strings of bounded length. Once the maximal string length is reached, the round concludes and a new round begins for the next subsequence of bounded length.

For authentication across rounds, the rounds must themselves be maintained in a prefix-authenticating data structure. This requires an awkward nesting of prefix authentication schemes that is overall less efficient than authenticating the full string without subdividing it into rounds.

Prefix authentication for strings of bounded length is an easier problem than prefix authentication for strings of arbitrary length. Hence, we need to adapt the timestamping schemes to the more general setting, and this adaptation results in worse positional certificate sizes than the original publications report. The original publications do not account for the cost of inter-round authentication, which is why we do our own complexity analyses and arrive at worse bounds.
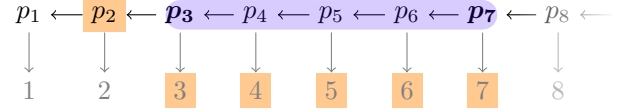

Figure 4: The linear linking scheme, highlighting certificate_vertices(3, 7) and certify(3, 7).

This adaptation also means that the proofs of optimality in the timestamping literature do not apply to our setting. While we believe our adaptations are faithful, more efficient solutions for round-less authentication can exist.

6.1 Trivial Schemes

The simplemost linking scheme is the *linear scheme* of [17]. Its underlying graph is a "Merkle linked-list" G_{lin}:

$$V_{lin} := \{p_n : n \in \mathbb{N}\} \cup \mathbb{N},$$
$$E_{lin} := \{(p_{n+1}, p_n) : n \in \mathbb{N}\} \cup \{(p_n, n) : n \in \mathbb{N}\},$$
$$G_{lin} := (V_{lin}, E_{lin}).$$

To use this graph as a linking scheme, define digest_vertex(n) := p_n, and define certificate_vertices(len_s, len_t) as the shortest path from digest_vertex(len_t) to digest_vertex(len_s) (see fig. 4 for a depiction). We use the same definitions of digest_vertex and certificate_vertices for all linking schemes in this section, unless specified otherwise.

Prefix certificates are of linear size in $len_t - len_s$. Certificate pools are of linear size in len_s, as they must contain the full path from digest_vertex(len_s) to digest_vertex(1): certificate_pool(len_s) := $G_{lin}[len_s]$. On the plus side, digest pools are of constant size, with digest_pool(len_s) := digest_vertex(len_s).

The *full linking scheme* is the other trivial scheme, with an edge from p_j to p_i for all $i < j$; the quadratic number of edges makes it unsuitable for any practical use.

6.2 Skip List Schemes

A simple but suboptimal way of interpolating between the two trivial schemes is to use a (contracted) deterministic skip list [35]. In addition to the edges of G_{lin}, also add an edge from p_n to p_{n-k} if n is divisible by k. The certificate pool of n is the out-neighborhood of the shortest path from $2^{\lceil \log_2(n) \rceil}$ to n and from n to 1. The digest pool is the out-neighborhood of the shortest path from n to 1. Figure 5 visualizes the construction.

This scheme is used by CHAINIAC [31], but it has prefix certificates of superlogarithmic size. Consider the vertex $n := 2^k$. It has k out-neighbors, all of which must occur in certificate_vertices($1, 2^k$). The second vertex of the shortest
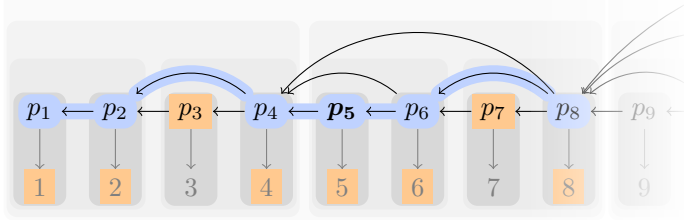
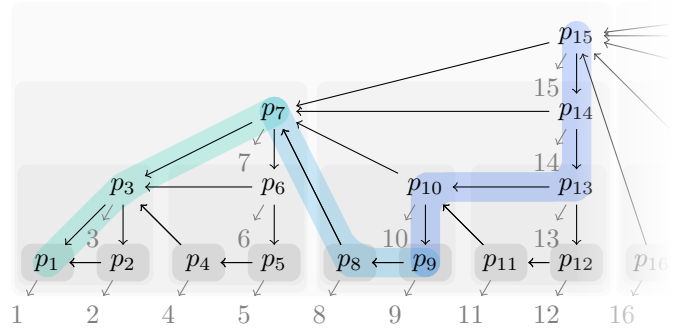Figure 5: The CHAINIAC scheme, highlighting certificate_pool(5) and its out-neighborhood.

Figure 6: The simple antimonotone binary linking scheme, highlighting certificate_pool(9), which consists of the paths from the next vertebra to p_9, from p_9 to the previous vertebra, and from the previous vertebra to p_1.

path from 2^k to 1 is 2^{k-1}, whose out-neighborhood contains $k-1$ vertices. Iterating this argument yields a certificate size of $\Sigma_{i \leq k} i \in O(k^2) = O(\log(n)^2)$. While certificate pools have logarithmic size, their out-neighborhoods do not.

Blibech and Gabillon [4] also propose a scheme based on skip lists, but their construction relies on the notion of timestamping rounds: unlike the timestamping schemes we generalize next, their scheme gives dedicated treatment to the last vertex of each timestamping round, and it can only provide prefix certificates for items whose timestamping rounds have been concluded. Hence, it is not a prefix authentication scheme according to our definition, as it is not applicable to authenticating sequences of unbounded length.

6.3 Antimonotone Binary Schemes

The *simple antimonotone binary linking scheme* [7] achieves logarithmic certificate pools by augmenting the linear scheme with only one additional outgoing edge per vertex. The additional edge for vertex p_n goes to $p_{f_2(n)}$, with $f_2(n)$ defined as follows:

$$f_2(n) := \begin{cases} n - (2^{k-1} + 1) & \text{if } n = 2^k - 1, k \in \mathbb{N} \\ n - 2^{g(n)} & \text{otherwise} \end{cases}$$

$$g(n) := \begin{cases} k & \text{if } n = 2^k - 1, k \in \mathbb{N} \\ g(n - (2^{k-1} - 1)) & \text{if } 2^{k-1} - 1 < n < 2^k - 1, k \in \mathbb{N}. \end{cases}$$

Observe that for all $n < m$ we have that $f_2(n) \geq f_2(m)$, hence the *antimonotone* in the name.

We denote the resulting graph as $G_{ls2} := (V_{lin}, E_{lin} \cup \{(p_n, p_{f_2(n)}) : n \in N^{\geq 2}\})$, fig. 6 shows an excerpt. For any n, we say it belongs to *generation* $\lfloor \log_2(n) \rfloor$. We say $p_{2^{t+1}-1}$ is the *vertebra* of generation t, and $\bigcup_{k \leq t} \{p_{2^{k+1}-1}\}$ is the *spine* of generation t.

Let n a number of generation t. Then, the union of the shortest paths from the vertebra of t to p_n, from p_n to the vertebra of $t-1$, and from that vertebra to p_1 (the latter two paths together form the shortest path from p_n to p_1) is a certificate pool for n [7]. We proceed with a proof sketch for the size of the corresponding positional certificates.

Observe the recursive structure of G_{ls2}: the graph of the first $t+1$ generations consist of two copies of the graph of the first t generations — with the outgoing edges of the spine of the copy being replaced with edges to the (original) vertebra of generation t — and a new vertebra $p_{2^{(t+1)+1}-1}$. This recursive structure enables convenient inductive reasoning based on the generation of a vertex.

Further observe that every vertex is either a vertebra vertex for some generation t, or a (transitive) copy of a vertebra vertex for some generation t. In both cases, we say that the vertex is of *order t*.

For n of generation t, the shortest path from the vertebra of t to the vertebra of $t-1$ via p_n is of maximal size if p_n is of order 0. The path consists of vertices of decrementing order from the vertebra of t to p_n, followed by vertices of incrementing order up to the vertebra of $t-1$. Consequently, all order 0 vertices of the same generation yield certficate pools of the same size: $2t-1$. A proper proof would perform induction on t: removing the vertebrae of t and $t-1$ from the path yields a path isomorphic to that of $n-2^t-1$, which is of generation $t-1$.

The out-neighborhood of this path has at least the same size, because every p_n has an edge to n. More tricky are the edges to other p_m; we need to count the number of such edges that lead outside the path. The edges corresponding to f_2 never do so, this would immediately contradict antimonotonicity. It remains to consider the edges of the form (p_n, p_{n-1}).

For vertices of generation at most 2, all these edges are part of the path. For vertices of any other generation $t+1$, there are $t-2$ such edges leading outside the path, as can be seen inductively: if the successor of vertex p_{2^t-3} on the path is not its numeric predecessor, the number of predecessor edges outside the path increases by one compared to the path for the previous generation. If the successor of vertex p_{2^t-3} on the path *is* its numeric predecessor, then $p_{f_2(2^t-3)}$ is part of the path but its predecessor is not, again contributing exactly one additional edge to a path of the previous generation (see

8

fig. 7).

For the full positional certificate of n of generation t, it remains to add the size of the out-neighborhood of the shortest path from the vertebra of $t-1$ to p_1. This path has t vertices, each contributing two vertices to the out-neighborhood (p_{m-1} and m), except for p_1, which has only a single out-neighbor. Adding everything up (and accounting for the double-counting of p_{t-1}) yields the positional certificate size of $(5 \cdot \lfloor \log_2(n) \rfloor - 3) \cdot k$, where k is the size of an individual hash.

The shortest path from p_n to p_1 is of logarithmic size and it can serve as a digest pool (in fact, antimonotonicity implies that this shortest path is a digest pool for *every* antimonotone scheme). The observation that the graph has no edge which jumps over any vertebra can be used to further shrink the digest pool to the shortest path from p_n to the vertebra of the previous generation.

The *optimal antimonotone binary linking scheme* [5] extends the linear scheme with a slightly different, antimonotone function $f_3(n)$, to obtain the graph $G_{ls3} := (V_{lin}, E_{lin} \cup \{(p_n, p_{(n)}) : n \in N^{\geq 2}\})$ (fig. 8):

$$f_3(n) := \begin{cases} n - (3^{k-1} + 1) & \text{if } n = \frac{3^k - 1}{2}, k \in \mathbb{N} \\ n - (\frac{3^{h(n)} - 1}{2} + 1) & \text{otherwise} \end{cases}$$

$$h(n) := \begin{cases} k & \text{if } n = \frac{3^k - 1}{2}, k \in \mathbb{N} \\ h(n - \frac{3^{k-1} - 1}{2}) & \text{if } \frac{3^{k-1} - 1}{2} < n < \frac{3^k - 1}{2}, k \in \mathbb{N} \end{cases}$$

Certificate pools (and their analysis) take the same shape as those of the simple antimonotone scheme, but with different generations and vertebra: the generation of n in the optimal scheme is $\lfloor \log_3(2n) \rfloor$, the vertebra of generation t is $\frac{3^{t+1} - 1}{2}$.

For n of generation t, inductive arguments analogous to to those of the simple antimonotone scheme yield a maximal size of $3t + 1$ for the path from the vertebra of t to the vertebra of $t-1$ via p_n, and $2t - 3$ additional vertices for the out-neighborhood. The shortest path from the vertebra of $t-1$ to p_1 contributes another $2t - 1$ vertices to the positional certificate, yielding the total size of $(7 \cdot \lfloor \log_3(2n) \rfloor - 4) \cdot k$ (again accounting for the double-counting of p_{t-1}). This is more efficient than the simple antimonotone scheme; for all $n \geq 128$ we have $(7 \cdot \lfloor \log_3(2n) \rfloor - 4) \cdot k < (5 \cdot \lfloor \log_2(n) \rfloor - 3) \cdot k$.

The shortest path from p_n to the vertebra of the previous generation can again serve as a digest pool. In the optimal antimonotone scheme, this path contains redundancies however. For example, consider p_{24} in fig. 8: none of the labels of p_{23}, p_{22}, or p_{17} are directly involved in the computation of the label of any greater vertex. For the digest pool of n, it suffices to take the p_m with maximal m of each order (with $m \leq n$ and for a maximal order of the generation of n).

We would like to point the interested reader to the elegant characterization of all antimonotone binary graphs [5] that forms the basis of the optimality proof for the optimal antimonotone scheme amongst all antimonotone binary schemes in the setting of prefix authentication for strings of bounded length. All antimonotone binary graphs can be constructed from a graph product operation $\otimes$, starting from the trivial graph G_1. The antimotone product is an efficient way of thinking about the antimonotone schemes; the rather intimidating functions of natural numbers to describe the two schemes we presented turn into neat, immediately related one-liners: $G_{simple}^{i+1} := G_{simple}^i \otimes G_{simple}^i \otimes G_1$ and $G_{opt}^{i+1} := G_{opt}^i \otimes G_{opt}^i \otimes G_{opt}^i \otimes G_1$.

6.4 Merkle Trees

Whereas the schemes we considered so far are extensions of G_{lin}, the remaining schemes utilize Merkle trees. To unify their presentation, we first define the infinite Merkle tree G_{tree} on which they build (fig. 9) in isolation:

$$V_{tree} := \{(n, k) : n \in \mathbb{N}, k \in \mathbb{N}_0 \text{ and } 2^k \mid n\},$$
$$E_{tree} := \{((n_0, k+1), (n_1, k)) : n_0 = n_1 \text{ or } n_0 = n_1 + 2^k\}$$
$$\cup \{((i, 0), i) : i \in \mathbb{N}\},$$
$$G_{tree} := (V_{tree}, E_{tree}).$$

We can unify parts of the complexity analysis of the remaining schemes by analyzing G_{tree}. We first define the forest that corresponds to the first n numbers: $G_{tree}[n] := G_{tree}[V_{tree}^{\leq n} \cup \{(i, k) : i \leq n\}]$. $G_{tree}[n]$ has at most $3n$ vertices for every n, but $G_{tree}[n] - G_{tree}[n-1]$ can have up to $\lceil \log_2(n) \rceil$ vertices. Unlike the schemes we have seen so far, all schemes based on G_{tree} thus require a non-constant amount of information for a single sequence item in the worst case.

We further define $\text{next\_root}(n) := (2^{\lceil \log_2(n) \rceil}, \lceil \log_2(n) \rceil)$, the root of the smallest complete subtree to contain both 1 and n, and $\text{next\_power}(n) := (2^{\lceil \log_2(n) \rceil}, 0)$.

The complexity analysis of several schemes depends on the number of roots in $G_{tree}[n]$. Observe that the number of leaves of every tree in $G_{tree}[n]$ is a power of two, and observe further that the trees in $G_{tree}[n]$ all have different, strictly decreasing sizes. Every power of two less than n occurs either once or not at all. In other words, the trees of $G_{tree}[n]$ correspond directly to the binary representation of n.

6.5 Threaded Authentication Trees

We now turn to the first construction to use Merkle trees, the *threaded authentication trees* [8]. Threaded authentication trees start from G_{tree} and then add edges from every $(n, 0)$ to the roots of the trees of $G_{tree}[n]$, yielding a linking scheme
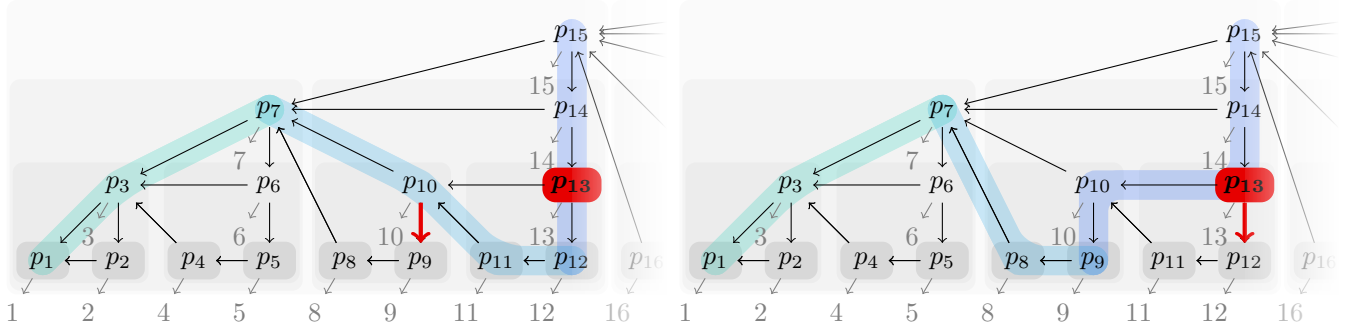
Figure 7: Visualizing the inductive step in counting the number of predecessor edges outside the certificate pool. Whether n of generation t is greater than or equal to $2^t + 2^{t-1} - 1$ (i.e., the numeric predecessor of p_{2^t-3} is part of the certificate pool) or not, a single edge more leads outside the certificate pool than for a number of generation $t-1$. The graphic shows the concrete case of $t := 3$ with $n := 12$ on the left and $n := 9$ right. The crucial vertex is p_{13}.
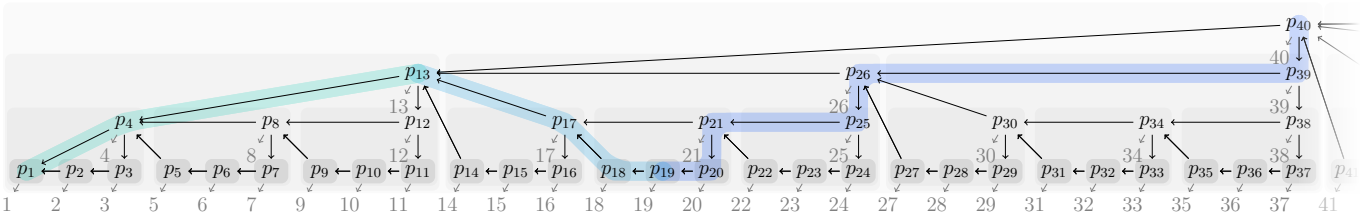

Figure 8: The optimal antimonotone binary linking scheme, highlighting certificate_pool(19), which consists of the paths from the next vertebra to $p_1 9$, from $p_1 9$ to the previous vertebra, and from the previous vertebra to p_1.

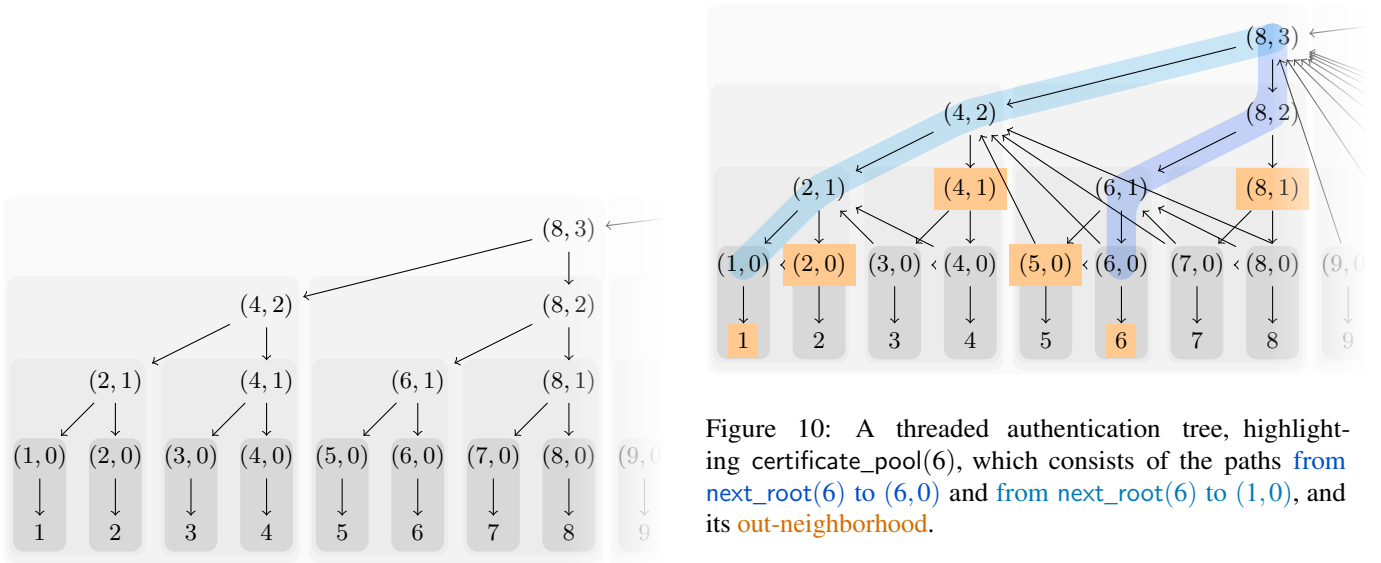
Figure 9: The start of the infinite Merkle tree G_{tree}.

Figure 10: A threaded authentication tree, highlighting certificate_pool(6), which consists of the paths from next_root(6) to (6,0) and from next_root(6) to (1,0), and its out-neighborhood.

with digest_vertex$(n) := (n, 0)$. The certificate pool of n is the union of the shortest path from next_root(n) to $(n, 0)$ and the shortest path from next_root(n) to $(1, 0)$. See fig. 10 for an example.

This definition of certificate pools yields positional certificates of size $2 \cdot \lceil \log_2(n) \rceil \cdot k$, where k is the size of a single hash. This is almost twice as much as in the setting
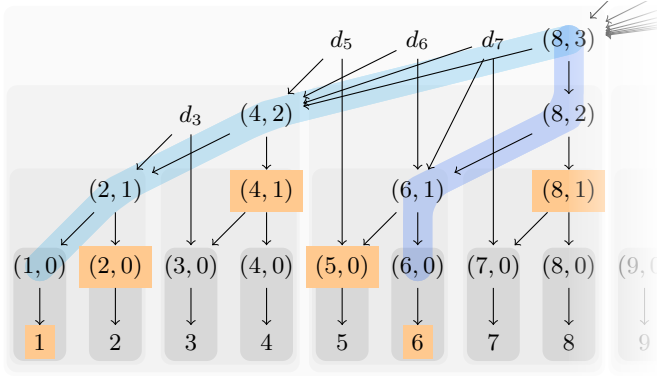
Figure 11: A hypercore, highlighting certificate_pool(6), which consists of the paths from next_root(6) to (6, 0) and from next_root(6) to (1, 0), and its out-neighborhood.

with rounds of known size, but it still outperforms the antimonotone schemes and is optimal among all schemes we survey. But unlike the antimonotone schemes, the underlying graph is of super-linear size, $G_{tat}[n]$ has $O(n \log(n))$ edges in the worst case:

Assume that $n = 2^k$. Remember that $(i, 0)$ has an outgoing edge for every 1 in the binary representation of i. The numbers from 2^{k-1} to 2^k (exclusively) are exactly the $\frac{n}{2}$ k-bit numbers. The total number of 1 bits amongst them — that is, the number of outgoing edges of the vertices $(\frac{n}{2}, 0), (\frac{n}{2} + 1, 0), \cdots, (n - 1, 0)$ — is $\frac{n}{2} \cdot \frac{k}{2} \in O(n \cdot k) = O(n \log(n))$.

As a consequence, checking a prefix certificate for two sequences of lengths $len_n < len_t$ can take time in $O(\log(len_t) \cdot \log(\log(len_t)))$, despite the size of the certificate being in $O(\log(len_t))$. This is asymptotically slower than checking certificates for antimonotone schemes. The original presentation of threaded authentication trees focuses on certificate size only and does not mention this flaw.

As the outgoing edges of any vertex (n, h) go to a root of a tree in $G_{tree}[n]$, these roots give a digest pool for n. Since these roots correspond to binary digits, the size of the digest pool of n is at most $\log_2(n)$.

6.6   Hypercore

Whereas threaded authentication trees turn G_{tree} into a linking scheme, we now turn to approaches that turn G_{tree} into more general TPASs. All these approaches share the insight that when $G_{tree}[n]$ has a single root, that root can serve as a digest vertex for a sequence of n items. But G_{tree} has no possible digest vertices for sequences of any other length.

Hypercore [33] takes a direct approach to augmenting G_{tree} so that there is a digest vertex for every n. For every n that is not a power of two, add a vertex d_n with an outgoing edge to every root of $G_{tree}[n]$ to obtain G_{hyper} (fig. 11).

Then define digest_vertex(n) as d_n, or the root of $G_{tree}[n]$ if

n is a power of two. commit(n) is then simply the set of roots of $G_{hyper}[n]$, a set which label-determines digest_vertex(n) by construction. Using these definitions of digest_vertex and commit leaves exactly one valid choice for defining certificate_vertices(len_s, len_t): the unique family of paths that start in digest_vertex(len_t) and end "just before" commit(len_t).

The certificate pool of n consists of the union of the path from next_root(n) to $(n, 0)$ and the path from next_root(n) to $(1, 0)$, just like with threaded authentication trees. Despite the different underlying graphs, the out-neighborhoods (and hence the sizes of positional certificates) are identical for threaded authentication trees and hypercores.

To see why the two paths yield valid certificate pools for hypercores, first observe that the out-neighborhood of the path from digest_vertex(n) to $(n, 0)$ is a subset of the out-neighborhood of the path from next_root(n) to $(n, 0)$.

Now consider any two numbers $len_s < len_t$. If next_root(len_s) $\neq$ next_root(len_t), then the path from next_root(len_t) to $(1, 0)$ contains next_root(len_s). And the out-neighborhood of the path from next_root(len_s) to $(len_s, 0)$ contains commit(len_s), so together they contain certificate_vertices(len_s, len_t).

In the other case of next_root(len_s) = next_root(len_t), both len_s and len_t lie in the complete subtree that contains the leaves $2^{\lceil \log_2(len_s) \rceil - 1} + 1$ to $2^{\lceil \log_2(len_s) \rceil}$. This tree is isomorphic to the tree with the first $2^{\lceil \log_2(len_s) \rceil - 1}$ leaves, so certificate_pool(len_s, len_t) is a valid certificate pool exactly if certificate_pool($len_s - 2^{\lceil \log_2(len_s) \rceil - 1}, len_t - 2^{\lceil \log_2(len_s) \rceil - 1}$) is a valid certificate pool. Hence, validity follows by induction — the base case is $len_s = 1$ and $len_t = 2$, for which the union of the certificate paths does contain certificate_vertices($1, 2$).

The roots of $G_{hyper}[n]$ are a digest pool for n; The labels of the roots of $G_{hyper}[n + 1]$ can be computed from the labels of the roots of $G_{hyper}[n]$ together with sequence item $n + 1$.

Like threaded authentication trees, hypercores are of super-linear size: $G_{hyper}[n]$ has $O(n \cdot \log(n))$ edges. But the number of edges within a prefix certificate is linear in the size of the certificate, so certificate validation lies in $O(\log(n))$.

Still, the super-linear size means that creating a sequence of length n step-by-step takes $O(n \log(n))$ time, whereas the antimonotone schemes only need $O(n)$ time. Furthermore, hypercores have identifiers of up to logarithmic size, unlike the constant-sized identifiers of antimonotone schemes or threaded authentication trees.

6.7   Transparency Logs

The *transparency log* construction [20] creates digest vertices for every n by iteratively adding a parent vertex to the roots of the two smallest trees in $G_{tree}[n]$ until there is a single root (fig. 12). This root then serves as the digest of n.

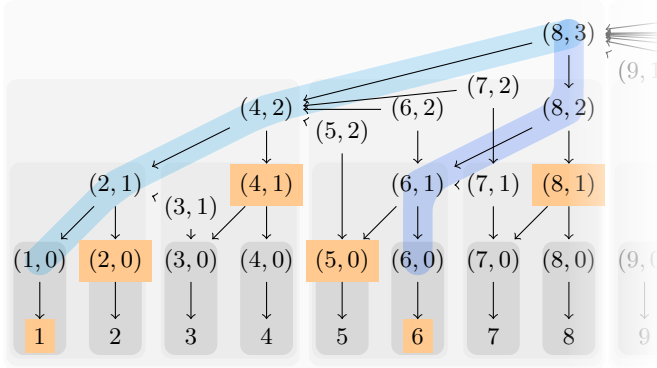Notice that contracting the newly created vertices for the same n yields exactly the underlying graph of the hypercore

Figure 12: A certificate transparency log, highlighting certificate_pool(6), which consists of the paths from next_root(6) to (6,0) and from next_root(6) to (1,0), and its out-neighborhood.

scheme. The certificate transparency scheme is ultimately a deterministic (but essentially arbitrary) way of subdividing the digest vertices of hypercore until every vertex has at most two outgoing edges. Consequently, commit(n) is identical to that of hypercore, and certificate_vertices(len_s, len_t) is again the unique family of paths that start in digest_vertex(len_t) and end "just before" commit(len_t).

Unlike hypercore, vertices in the certificate transparency scheme have a constant-bounded out-degree (two). The price is a super-linear number of vertices and the hashing of twice as many bits during certificate validation as hypercore. Otherwise, the same complexity analyses apply to both schemes.

In the context of certificate transparency, we want to point out that identifier sizes can matter outside timestamping. After submitting an entry to a certificate transparency log, the log operator replies[3] with the new *signed tree head* (*digest* in our terminology) and an *inclusion proof* (an *identifier* in our terminology). Using a scheme with constant-sized identifiers would be an asymptotic improvement over the certificate transparency scheme for this operation.

Five years prior to the publication of the certificate transparency scheme, Crosby and Wallach [11] presented a highly similar scheme: whereas certificate transparency creates the smallest possible binary trees that contain all trees of $G_{tree}[n]$, Crosby and Wallach's scheme also ensures that the vertices corresponding to sequence items all have the same height in the tree.

This makes the resulting graphs slightly larger supergraphs of the certificate transparency graphs, without any actual advantages. In hindsight, we can hence recommend to disregard Crosby and Wallach's scheme, while still appreciating that theirs is the first published non-linking scheme to solve prefix authentication.

---

[3]Setting aside CT-specific optimization details such as deliberate merge delays.

6.8 Summary

Prior presentation of all the schemes we surveyed has focused exclusively on prefix or positional certificate sizes. From this limited perspective, threaded authentication trees, hypercores and certificate transparency logs are equivalent, and antimonotone schemes are inferior.

The picture drastically changes when taking into account the other complexity metrics of prefix authentication schemes. All three schemes with minimal certificates have a super-linear total size. Threaded authentication trees have smaller identifiers than hypercore and transparency logs, but suffer from super-logarithmic verification times. The scheme of Blibech and Gabillon [4] does achieve the same certificate sizes with a linearly-sized underlying graph, but it does not generalize to the setting of unbounded sequence lengths.

Table 1 summarizes the results of our complexity analyses of the presented schemes.

7   Conclusion

Generalizing from secure time stamping and logging to prefix authentication has allowed us to transfer knowledge between results that have not been connected so far. The class of *transitive prefix authentication schemes* serves as a tool to compactly and efficiently present and analyze prior results. The nuanced analysis shows that no existing approach is strictly superior to any other. We hope that future system designs will take into account all complexity criteria of prefix authentication schemes rather than latching on the first scheme with sub-linear prefix certificates that they lay eyes upon.

The main questions we leave open are questions of optimality. While threaded authentication trees have provably minimal positional certificates amongst linking schemes in a timestamping setting with rounds of known length, we did not transfer the optimality result to the setting of prefix authentication *without* rounds. Similarly, we do not know whether the optimal antimonotone scheme for rounds of bounded length remains optimal amongst antimonotone schemes for prefix authentication without rounds.

Another open question is whether the optimal positional certificate sizes amongst *linking* schemes are optimal amongst all *transitive prefix authentication* schemes.

Our complexity analyses further surface a natural design challenge: that of finding a transitive prefix authentication scheme that achieves positional certificates of size $2 \cdot \lceil \log_2(n) \rceil \cdot k$ while having an underlying graph of linear size. Such a scheme would strictly outperform threaded authentication trees, hypercore, and certificate transparency logs.

Finally, we would like to emphasize again some of the limitations of prefix authentication schemes. First, the byzanthine fault-tolerant distributed "append-only log" does not exist, as reacting to forks adds more expressivity to the abstract data type than just an append-operation. And second, "proac-

	Linear	Full	Skiplist	Simple Antimonotone
Positional Certificate	$n \cdot k$	$n \cdot k$	$O(\log(n)^2) \cdot k$	$(5 \cdot \lfloor \log_2(n) \rfloor - 3) \cdot k$
Certificate Validation	$O(certsize)$	$O(certsize^2)$	$O(certsize)$	$O(certsize)$
Edges Amortized	$O(n)$	$O(n^2)$	$O(n)$	$O(n)$
Edges Worst Case	$O(1)$	$O(n)$	$O(\log(n))$	$O(1)$
Vertices Amortized	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Vertices Worst Case	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Identifier Amortized	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Identifier Worst Case	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Digest Pool	1	n	$\lfloor \log_2(n) \rfloor$	$\lfloor \log_2(n) \rfloor$
	Optimal Antimonotone	Threaded Authentication	Hypercore	Transparency Log
Position Certificate	$(7 \cdot \lfloor \log_3(2n) \rfloor - 4) \cdot k$	$2 \cdot \lceil \log_2(n) \rceil \cdot k$	$2 \cdot \lceil \log_2(n) \rceil \cdot k$	$2 \cdot \lceil \log_2(n) \rceil \cdot k$
Certificate Validation	$O(certsize)$	$O(certsize \cdot \log(certsize))$	$O(certsize)$	$O(certsize)$
Edges Amortized	$O(n)$	$O(n \cdot \log(n))$	$O(n \cdot \log(n))$	$O(n \cdot \log(n))$
Edges Worst Case	$O(1)$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$
Vertices Amortized	$O(n)$	$O(n)$	$O(n)$	$O(n \cdot \log(n))$
Vertices Worst Case	$O(1)$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$
Identifier Amortized	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Identifier Worst Case	$O(1)$	$O(1)$	$O(\log(n))$	$O(\log(n))$
Digest Pool	$\lfloor \log_3(2n) \rfloor$	$\lfloor \log_2(n) \rfloor$	$\lfloor \log_2(n) \rfloor$	$\lfloor \log_2(n) \rfloor$

Table 1: Summary of the complexity analyses of all presented schemes. The number of edges and vertices determines how long it takes to create a sequence of n items. Amortized complexities report the metric for $G[n]$, worst-case complexities report the metric for $G[n] \setminus G[n-1]$.

tive" fork detection by placing logs onto a blockchain only shifts the problem to that of detecting forks in the blockchain, which might remain undetected for just as long as forks in a log when dealing with an equally powerful adversary (as it is literally the exact same problem).

The universe is cold and dark when it comes to distributed systems, but pretending otherwise always does more harm than good.

References

[1] Mustafa Al-Bassam and Sarah Meiklejohn. Contour: A practical system for binary transparency. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 94–110. Springer, 2018. https://arxiv.org/pdf/1712.08427.pdf.

[2] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. Arpki: Attack resilient public-key infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 382–393, 2014. https://cispa.saarland/group/cremers/downloads/papers/ccsfp200s-cremersA.pdf.

[3] Dave Bayer, Stuart Haber, and W Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences Ii*, pages 329–334. Springer, 1993. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=fcc58b43fe133e98025bc616fbddd96393ae48c7.

[4] Kaouthar Blibech and Alban Gabillon. A new timestamping scheme based on skip lists. In *Computational Science and Its Applications-ICCSA 2006: International Conference, Glasgow, UK, May 8-11, 2006, Proceedings, Part III 6*, pages 395–405. Springer, 2006. https://www.researchgate.net/profile/Alban-Gabillon/publication/221432970_A_New_Timestamping_Scheme_Based_on_Skip_Lists/links/0deec52aff664d7605000000/A-New-Timestamping-Scheme-Based-on-Skip-Lists.pdf.

[5] Ahto Buldas and Peeter Laud. New linking schemes for digital time-stamping. In *ICISC*, volume 98, pages 3–14. Citeseer, 1998. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d3a005fb546ff78abc6ee453af4ee91aa6267c50.

[6] Ahto Buldas, Peeter Laud, and Helger Lipmaa. Eliminating counterevidence with applications to accountable certificate management. *Journal of Computer Security*, 10(3):273–296, 2002. https://research.cyber.ee/~peeter/research/JCS161.pdf.

[7] Ahto Buldas, Peeter Laud, Helger Lipmaa, and Jan Villemson. Time-stamping with binary linking schemes.

In *Annual International Cryptology Conference*, pages 486–501. Springer, 1998. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=f2390ec334bf99cb3d532bc16e05b6b201ad7115.

[8] Ahto Buldas, Helger Lipmaa, and Berry Schoenmakers. Optimally efficient accountable time-stamping. In *International workshop on public key cryptography*, pages 293–305. Springer, 2000. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=aa47957c6bc0e2c19b39aa644cb6b2e0c1defc83.

[9] Melissa Chase and Sarah Meiklejohn. Transparency overlays and applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 168–179, 2016. https://discovery.ucl.ac.uk/id/eprint/10055892/1/transparency.pdf.

[10] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 415–423. IEEE, 2015. https://arxiv.org/pdf/1511.01514.pdf.

[11] Scott A Crosby and Dan S Wallach. Efficient data structures for tamper-evident logging. In *USENIX security symposium*, pages 317–334, 2009. https://www.usenix.org/legacy/event/sec09/tech/full_papers/crosby.pdf.

[12] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, 1987. https://dl.acm.org/doi/pdf/10.1145/41840.41841.

[13] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure logging schemes and certificate transparency. In *European Symposium on Research in Computer Security*, pages 140–158. Springer, 2016. https://eprint.iacr.org/2016/452.pdf.

[14] Adam Eijdenberg, Ben Laurie, and Al Cutter. Verifiable data structures. *Google Research, Tech. Rep*, 2015. https://continusec.com/static/VerifiableDataStructures.pdf.

[15] Sascha Fahl, Sergej Dechand, Henning Perl, Felix Fischer, Jaromir Smrcek, and Matthew Smith. Hey, nsa: Stay away from my market! future proofing app markets against powerful attackers. In *proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1143–1155, 2014. https://teamusec.de/pdf/conf-ccs-FahlDPFSS14.pdf.

[16] Michael T Goodrich and Roberto Tamassia. Efficient authenticated dictionaries with skip lists and commutative hashing. Technical report, Technical Report, Johns Hopkins Information Security Institute, 2000. https://cs.brown.edu/cgc/stms/papers/hashskip.pdf.

[17] Stuart Haber and W Scott Stornetta. How to time-stamp a digital document. In *Conference on the Theory and Application of Cryptography*, pages 437–455. Springer, 1990. https://link.springer.com/content/pdf/10.1007/BF00196791.pdf.

[18] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil Gligor. Accountable key infrastructure (aki) a proposal for a public-key validation infrastructure. In *Proceedings of the 22nd international conference on World Wide Web*, pages 679–690, 2013. https://dl.acm.org/doi/abs/10.1145/2488388.2488448.

[19] Murat Yasin Kubilay, Mehmet Sabir Kiraz, and Hacı Ali Mantar. Certledger: A new pki model with certificate transparency based on blockchain. *Computers & Security*, 85:333–352, 2019. https://arxiv.org/pdf/1806.03914.pdf.

[20] Ben Laurie. Certificate transparency. *Communications of the ACM*, 57(10):40–46, 2014. https://dl.acm.org/doi/fullHtml/10.1145/2659897.

[21] Ben Laurie. Certificate transparency: Public, verifiable, append-only logs. *Queue*, 12(8):10–19, 2014. https://dl.acm.org/doi/pdf/10.1145/2668152.2668154.

[22] Ben Laurie and Emilia Kasper. Revocation transparency. *Google Research, September*, 33, 2012. http://www.links.org/files/RevocationTransparency.pdf.

[23] Ben Laurie, Adam Langley, Emilia Kasper, Eran Messeri, and Rob Stradling. Certificate Transparency Version 2.0. RFC 9162, December 2021. https://www.rfc-editor.org/info/rfc9162.

[24] Hemi Leibowitz, Haitham Ghalwash, Ewa Syta, and Amir Herzberg. Ctng: Secure certificate and revocation transparency. *Cryptology ePrint Archive*, 2021. https://eprint.iacr.org/2021/818.pdf.

[25] Jinyuan Li, Maxwell N Krohn, David Mazieres, and Dennis E Shasha. Secure untrusted data repository (sundr). In *Osdi*, volume 4, pages 9–9, 2004. https://www.usenix.org/legacy/event/osdi04/tech/full_papers/li_j/li_j.pdf.

[26] DSV Madala, Mahabir Prasad Jhanwar, and Anupam Chattopadhyay. Certificate transparency using blockchain. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 71–80. IEEE, 2018. https://eprint.iacr.org/2018/1232.pdf.

[27] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. {CONIKS}: Bringing key transparency to end users. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 383–398, 2015. https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-melara.pdf.

[28] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018. http://labit501.upct.es/~fburrull/docencia/SeguridadEnRedes/old/teoria/bibliography/HandbookOfAppliedCryptography_AMenezes.pdf.

[29] Ralph C Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1989. https://link.springer.com/content/pdf/10.1007/0-387-34805-0_21.pdf.

[30] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. *IEEE Journal on selected areas in communications*, 18(4):561–570, 2000. https://www.usenix.org/legacy/publications/library/proceedings/sec98/full_papers/nissim/nissim.pdf.

[31] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. {CHAINIAC}: Proactive {Software-Update} transparency via collectively signed skipchains and verified builds. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1271–1287, 2017. https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-nikitin.pdf.

[32] Linus Nordberg, Daniel Kahn Gillmor, and Tom Ritter. Gossiping in CT. Internet-Draft draft-ietf-trans-gossip-05, Internet Engineering Task Force, January 2018. https://datatracker.ietf.org/doc/draft-ietf-trans-gossip/05/.

[33] Maxwell Ogden, Karissa McKelvey, Mathias Buus Madsen, et al. Dat — distributed dataset synchronization and versioning. *Open Science Framework*, 10, 2017. https://terrymarine.com/wp-content/uploads/2017/07/724d7267d90052778b0530807512474b.pdf.

[34] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Authenticated hash tables. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 437–448, 2008. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0556a7a0ceec16986a1817333de62149548b95ad.

[35] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990. https://dl.acm.org/doi/pdf/10.1145/78973.78977.

[36] Tobias Pulls and Roel Peeters. Balloon: A forward-secure append-only persistent authenticated data structure. In *European Symposium on Research in Computer Security*, pages 622–641. Springer, 2015. https://eprint.iacr.org/2015/007.pdf.

[37] Tobias Pulls, Roel Peeters, and Karel Wouters. Distributed privacy-preserving transparency logging. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 83–94, 2013. https://www.esat.kuleuven.be/cosic/publications/article-2373.pdf.

[38] Mark D Ryan. Enhanced certificate transparency and end-to-end encrypted mail. *Cryptology ePrint Archive*, 2013. https://eprint.iacr.org/2013/595.pdf.

[39] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security (TISSEC)*, 2(2):159–176, 1999. https://dl.acm.org/doi/pdf/10.1145/317087.317089.

[40] Abhishek Singh, Binanda Sengupta, and Sushmita Ruj. Certificate transparency with enhancements and short proofs. In *Information Security and Privacy: 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3–5, 2017, Proceedings, Part II 22*, pages 381–389. Springer, 2017. https://arxiv.org/pdf/1704.04937.pdf.

[41] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities" honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 526–545. Ieee, 2016. https://arxiv.org/pdf/1503.08768.pdf.

[42] Roberto Tamassia. Authenticated data structures. In *European symposium on algorithms*, pages 2–5. Springer, 2003. https://hashingit.com/elements/research-resources/2003-Tamassia-ADS.pdf.

[43] Dominic Tarr, Erick Lavoie, Aljoscha Meyer, and Christian Tschudin. Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications. In *Proceedings of the 6th ACM conference on information-centric networking*, pages 1–11, 2019. https://dl.acm.org/doi/pdf/10.1145/3357150.3357396.

[44] Alin Tomescu, Vivek Bhupatiraju, Dimitrios Papadopoulos, Charalampos Papamanthou, Nikos Triandopoulos, and Srinivas Devadas. Transparency logs via append-only authenticated dictionaries. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1299–1316, 2019. https://dl.acm.org/doi/pdf/10.1145/3319535.3345652.

[45] Alin Tomescu and Srinivas Devadas. Catena: Efficient non-equivocation via bitcoin. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 393–409. IEEE, 2017. https://dspace.mit.edu/bitstream/handle/1721.1/137544/catena.pdf?sequence=2.

[46] Ze Wang, Jingqiang Lin, Quanwei Cai, Qiongxiao Wang, Daren Zha, and Jiwu Jing. Blockchain-based certificate transparency and revocation transparency. *IEEE Transactions on Dependable and Secure Computing*, 2020. https://fc18.ifca.ai/bitcoin/papers/bitcoin18-final29.pdf.

[47] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001. https://faculty.math.illinois.edu/~west/igt/igtpref.ps.

[48] Jiangshan Yu, Vincent Cheval, and Mark Ryan. Dtki: A new formalized pki with verifiable trusted parties. *The Computer Journal*, 59(11):1695–1713, 2016. https://arxiv.org/pdf/1408.1023.pdf.

[49] Aydan R Yumerefendi and Jeffrey S Chase. Strong accountability for network storage. *ACM Transactions on Storage (TOS)*, 3(3):11–es, 2007. https://www.usenix.org/legacy/event/fast07/tech/full_papers/yumerefendi/yumerefendi.pdf.