Secure Data Structures

Thesis Proposal

Sam A. Markelon
University of Florida
smarkelon@ufl.edu

October 2024

1 Introduction

Data structures define representations of possibly dynamic (multi)sets, along with the operations that can be performed on this representation of the underlying data. Efficient data structures are crucial for designing efficient algorithms [1]. The development and analysis of data structures has largely been driven by operational concerns, e.g., efficiency, ease of deployment, support for broad application. Security concerns, on the other hand, have traditionally been afterthoughts (at best). However, recent research has highlighted that many widely-used data structures do not behave as expected when in the presence of adversaries that have the ability to control the data they represent. Further, complex protocols that have sophisticated security goals are increasingly using a variety of bespoke data structures as fundamental components of their design. Therefore, it is wise to begin applying the provable security paradigm to data structures themselves.

For instance, consider probabilistic data structures (PDS). They provide compact (sublinear) representations of potentially large collections of data and support a small set of queries that can be answered efficiently. Prime examples of such structures include the Bloom filter [2], the HyperLogLog [3], and the Count-min Sketch [4]. These space and (by extension) performance gains come at the expense of correctness. Specifically, PDS query responses are computed over the compact representation of the data, as opposed to the complete data. As a result, PDS query responses are only guaranteed to be *close* to the true answer with *large* probability, where *close* and *large* are typically functions of structure parameters (e.g., the representation size) and properties of the data. These guarantees are stated under the assumption that the data and the internal randomness of the PDS are independent. Informally, this is tantamount to assuming that the entire collection of data is (or can be) determined *before* any random choices are made by the PDS. For many PDS, this means before some number of hash functions are sampled, as the PDS operates deterministically after that. Recent works have begun to explore the impact on correctness guarantees for data that *may* depend upon the internal randomness of the structure, and the initial findings are negative.

Moreover, consider the class of data structures we refer to as *skipping data structures*. Unlike the probabilistic data structures we discussed earlier, this class of structure are not space-efficient (compact) and, in turn, give exact answers to queries. These data structures (e.g., hash tables, skip lists, and treaps) offer fast average-case runtime of their operations, but have worst-case runtime that is poor. They achieve this by using some form of randomness to determine the

representation of the underlying data collection. Recent research shows that adaptive adversaries are able to force worst-case runtime for these structure, often demonstrated by attacks on real-world systems. Therefore, instead of focusing on adversarial correctness as in the PDS section, we focus on preserving the expected run time of these structures with large probability in the presence of an adversary.

Lastly, we consider verifiable data structures. Unlike the previous two classes of data structures we consider, these structures are designed with specific and explicit security notions in mind. While verifiable (or authenticated) data structures have existed in the literature for decades [5], we are specifically interested in the use of verifiable data structures in the context of *key transparency systems*. Informally, key transparency systems are employed by end-to-end messaging systems with large user bases to automatically ensure users are getting the correct public key of another user they wish to communicate with. We explore a mismatch between the needs of large-scale key transparency systems, and the security properties that are captured in the traditional verifiable structure literature. We also explore how a novel privacy-preserving summary mechanism can be used to solve scalability issues with current deployments.

These results illustrate the need to rigorously analyze the security properties of certain classes of data structures, thus we put forth the following thesis statement:

1.1 Thesis Statement

The security properties of data structures have largely been an afterthought (at best). To formalize the behavior of these structures (and the upstream applications that use them) in adversarial environments, it is necessary to employ the provable security paradigm in their analysis.

2 Background

We now provide a brief summary of related work for the classes of data structures that we focus on.

2.1 Probabilistic Data Structures

2.1.1 Approximate Set Membership Data Structures

The first works to explore PDS in a provable security style focused on the Bloom filter [2]. The Bloom filter admits approximate set-membership queries. The structure is widely used in many computing contexts, such as databases [6], networking [7], distributed systems [8], and search [9].

Naor and Yogev were the first to consider settings in which inputs and queries may be chosen by an adaptive adversary and formally investigate attacks that can occur in such a setting [10]. Their results show that adversaries can find queries that are guaranteed to be false positive for a given instantiation of a filter and data collection. They formalized a notion of adversarial correctness for a modified Bloom filter structure of their own construction and provide a correctness bound for it. Clayton et al. [11] extend this work by considering stronger adversaries. They allow for the adversary to insert elements into the structure after the adversary has started to issue queries – that is, they consider a fully mutable setting. They find that the basic Bloom filter is vulnerable to adversarial manipulation, which can increase false positives to nearly 100%. To secure it, they recommend adding a unique salt in an immutable setup, or using a private representation, keyed hash functions, and insertion thresholds in a mutable setting. Further, they formalize a notion of

adversarial correctness that extends past only Bloom filters, also concretely analyzing the counting filter [12] and the Count-min sketch [4]. Filić et al. [13] further analyze the adversarial correctness of Bloom filters and Cuckoo filters (another approximate membership data structure) in a simulation style security notion. They reach similar conclusions to [11].

2.1.2 HyperLogLog

The HyperLogLog (HLL) [3] is a PDS that provides a compact representation of a set and can accurately approximate the number of distinct elements in the set (i.e., the set's cardinality). Patterson and Raynal [14] provide a provable security treatment of the HLL. They first present attacks which exploit the use of fixed and publicly computable hash function in the HLL to cause large cardinality estimate errors. They then show that by switching these hash functions for a secretly keyed primitive that (even in the setting where an adversary has complete access to the internal state of the structure) the structure remains secure in terms of conserving the non-adversarial correctness guarantees of the structure. Prior to this, Revirigeo and Ting provide attacks against the HLL in a model where the adversary has access to a "shadow" device that mirrors the structure that is being attacked [15]. Patterson and Raynal point out this setting in unrealistic, but nonetheless improve the attack in this model.

2.1.3 Compact Frequency Estimators

Compact frequency estimators are a class of PDS that compactly represent a collection of streaming data (usually modeled as a multiset), and provide approximately correct frequency estimates (that is, the number of times any particular element has appeared in the stream). Alternately, compact frequency estimators can be viewed as providing a compact representation of the frequency distribution of a particular data stream.

As previously stated, Clayton et al. were the first to examine compact frequency estimators from a provable security perspective [11]. They specifically examined the Count-min sketch and presented attacks that could cause large frequency estimation error when the internals state of the structure or the hash functions used by the structure were made available to the adversary. They were able to prove security of the structure when the internal state of the structure is kept private and a secretly keyed primitive was used in place of the usual hash functions. However, their defined adversarial goal was very conservative. Any fixed amount of frequency estimation error was considered a win for the adversary, rather than an accumulated error that surpassed that of the non-adversarial correctness guarantee. Further, their construction relied on a thresholding technique, in which the structure would not accept any more updates after a bounded number of insertions.

We continued on this line of work in our 2023 ACM CCS Paper *Compact Frequency Estimators in Adversarial Settings* [16]. We detail this work in Section 3.1. Further, we explore attacks against actual implementations of PDS in *Probabilistic Data Structures in the Wild: A Security Analysis of Redis* [17]. This work will soon be in submission, and we describe it in Section 3.2.

2.2 Skipping Data Structures

We take *skipping data structures* to mean non-compact (i.e., they are linear in the size of the collection they represent) structures that use some form of randomness to provide fast expected run times (sublinear) in their core operations. For instance, consider the hash table (with expected

constant run-time for insertion, deletion, and search) or the skip list (with expected $O(\log n)$ run-time for insertion, deletion, and search). However, in the case of the hash table, this average-case performance is based on the assumption that the data inserted into a hash table is independent of the (possibly random) choice of hash function used to map key-values pairs to buckets. In the case of the skip list, this average case performance is based on the assumption that random choices taken at insertion time are not manipulated by (targeted) subsequent deletion operations.

Many previous works have examined how to exploit a bad choice of hash functions to force operations to degrade to the worst-case performance, $O(n)$ where n is the total number of elements residing in the structure. Crosby and Wallach [18] investigate denial-of-service (DoS) attacks against a number of applications that internally use hash tables to process and store data. The authors show that by selecting input data such that all the elements hash to the same bucket, performance of the upstream application can be degraded. Their attacks rely on the use of weak (non-cryptographic) and fixed hash function. Specifically, they were able to cause the Bro network-intrusion detection system [19] to fail by overloading the system, causing it to drop all network traffic for a large period of time. Similarly, Klink and Walde [20] present attacks that caused web applications servers to use 99% of their CPU for prolonged lengths of time by only sending a single carefully crafted HTTP request. These attacks also exploited a bad choice of hash functions in the implementations of hash tables in many common programming languages (PHP, ASP.NET, Java, etc.) that caused worst-case performance on targeted web servers.

Follow-up work by Aumasson et al. [21] showed further vulnerabilities in many programming languages' default hash table implementations. This was done by analysis of the commonly used non-cryptographic MurmurHash2, MurmurHash3, and CityHash64 hash functions [22], leading to efficient algorithms for generating arbitrary multi-collisions in all cases. SipHash [23], a pseudorandom function designed to be fast for short-inputs, was proposed as an alternative that prevented these attacks and has been adopted by many of the affected programming languages. While this fix seems to work in practice [24], (to our knowledge) no formalization concerning the provable security of these secretly keyed hash tables exist.

A skip list [25] is a data structure that uses a randomized storage technique for efficient insertion, deletion, and search on an ordered sequence of elements. As opposed to balanced trees, skip lists are faster on real hardware, more space efficient, and less complex to implement, while possessing the same asymptotic average runtime ($O(\log n)$ for all operations). Upon an element's insertion into the structure, the element is assigned a random height from one up to some max height, with higher heights being increasingly less probable. This randomized height mechanism allows one to efficiently skip over a number of elements when conducting a search on the ordered data by starting a search at the maximum height and only traversing down heights (where a larger proportion of elements reside) as necessary.

The original skip list paper [25] notes that an adversarial user can force worst-case runtime for the operations of a skip list ($O(n)$ for all operation) if they had access to the heights of the elements in the structure. The attack is simple; an adversary only needs to delete any element with height greater than 1. This degenerates the list by flattening it and removing the possibility of any skips occurring when a search is performed (which is also the core suboperation for both insertion and deletion). Thus, it is crucial to keep the internal structure of a skip list private to prevent this attack. However, recent work by Nussbaum and Segal [26] show that the internal structure of a skip list can be discovered even if kept private through timing attacks. By issuing a series of search queries, an attacker is able to correlate the height of an element with the time taken to respond. Once the heights of elements contained in the structure are discovered, the attacker then simply

carries out the attack described in the original paper. The authors suggest a structure called a splay skip list, that randomizes the heights of the elements during a search query, to prevent this attack, but provide no formalization of its security.

In ongoing work called *Skipping Data Structures in Adversarial Environments*, we create a novel security model for skipping data structures based on conserving desirable properties of the representation that structure maintains in adversarial conditions. Further, we introduce provably secure and efficient constructions for hash tables and skip lists. To our knowledge, we are the first to provide a provable security style treatment to skipping data structures. We detail this work in Section 4.1.

2.3 Verifiable Data Structures and Key Transparency

Verifiable data structures assume a model in which a trusted source creates a data structure from some initial data collection, and publishes a commitment (or more generally, public verification information) to it. The structure is then given to a query-responding party, who is *not* assumed (by clients, anyway) to be honest. Clients can then make queries to this untrusted responder, and verify the validity of the response (with respect to the honestly generated representation) using the published commitment. This verifiable structure paradigm is useful when a data collection is being replicated over many responders (for, say, efficiency and security reasons) and these responders have incentives to be dishonest. Some applications where these structures have been employed include certificate revocation[27, 28, 29, 30] and downloading content from (untrusted) internet mirrors [31, 32].

The Merkle Tree [33, 34] is the ubiquitous example of a verifiable data structure. It provides efficient and secure verification of a potentially large collection of immutable data. A number of other works in the verifiable data structures space focus on making verifiable versions of specific immutable data structures (e.g., dictionaries, trees, graphs, etc.) [35, 36, 27, 37, 38]. There has also been work done on crafting verifiable data structures which allow for updates [36, 39, 40]. A formal abstraction for (mutable) verifiable data structures is given in [41], and a method for crafting verifiable data structures generically is presented in [42]. Lastly, there has been a line of work exploring zero-knowledge sets, which extended strong privacy guarantees to verifiable structures [43, 44, 45, 46, 47, 48].

We are particularly interested in the use of verifiable data structures in the context of Key Transparency (KT) systems. KT systems address the challenges of public key distribution in end-to-end encrypted communication platforms and are crucial for preventing trivial *man-in-the-middle* attacks. This has traditionally been a cumbersome process that requires physical meetings between users that wish to authenticate themselves to one another. They make use of a privacy-preserving verifiable key directory data structure, that is concretely instantiated as an ordered append-only zero-knowledge set [49, 50].

We explore how to solve scalability challenges in KT deployments in ongoing work *Solving Scalability for Key Transparency Systems via a Privacy-Preserving Verifiable Summary Mechanism*. This is detailed in Section 4.2. We also explore a planned follow-up work that generalizes the verifiable summary mechanism, entitled *Compact, Private, and Verifiable Data Structures*, in Section 4.3.

3 Completed Work

3.1 Compact Frequency Estimators in Adversarial Environments

The following is a summary of our work presented at ACM CCS 2023 [16].

In this work, we focus on PDS that can be used to estimate the number of times any particular element x appears in a collection of data, i.e., the *frequency* of x. Such compact frequency estimators (CFEs) are commonly used in streaming settings, to identify elements with the largest frequencies — so-called *heavy hitters* or *elephants*. Finding extreme elements is important for network planning [51], network monitoring [52], recommendation systems [53], and approximate database queries [54], to name a few applications.

The Count-min Sketch (CMS) [4] and HeavyKeeper (HK) [55] structures are two CFEs that we consider, in detail. The CMS structure has been widely applied to a number of problems outlined above. Details on these applications are thoroughly examined in the survey paper by Sigurleifsson et al. [56]. The HK structure is the CFE of choice in the RedisBloom module [54], a component of the Redis database system [57].

Of particular interest to us is the 2019 ACM SIGSAC work of Clayton, Patton, and Shrimpton [11] that both furthers the adversarial analysis on Bloom filters and also presents a general model for analyzing probabilistic data structures for provable security. This paper gives a first look at the security of the Count-min sketch in adversarial environments. However, in this paper a very conservative security model for the CMS was used, which counted any overestimation of a particular element as an adversarial gain, rather than tying the security to the non-adaptive guarantees of the structure. Further, a thresholding mechanism is used to achieve security for the CMS, a solution which we deem untenable for real world uses of the CMS.

As is the case for other PDS, the accuracy guarantees for CFEs effectively assume that the data they represent were produced by a non-adaptive strategy. Our work explores the accuracy of CMS and HK estimates when the data is produced by *adaptive* adversarial strategies (i.e., adaptive attacks). We give explicit attacks that aim to make as-large-as-possible gaps between the estimated and true frequencies of data elements. We give concrete, not asymptotic, expressions for these gaps, in terms of specific adversarial resources (i.e., oracle queries), and support these expressions with experimental results. And our attacks fit within a well-defined "provable security"-style attack model that captures four adversarial access settings: whether the CFE representations are publicly exposed (at all times) or hidden from the adversary, and whether the internal hash functions are public (i.e., computable offline) or private (i.e., visible only, if at all, by online interaction with the structure).

In this work we draw explicit attention to the fact that probabilistic data structures, and in particular frequency estimators, were not designed with security in mind by presenting attacks that degrade the correctness of the query responses these structures provide.

Our findings are negative in all cases. No matter the combination of public and private, a well resourced adversary can force CMS and HK estimates to be arbitrarily far from the true frequency. As one example of what this means for larger systems, things that have never appeared in the stream can be made to look like heavy hitters (in the case of CMS), and legitimate heavy hitters can be made to disappear entirely (in the case of HK). This is somewhat surprising in the "private-private" setting, where the attack can only gain information about the structure and its operations via frequency estimate queries. Of course, there are differences in practice: when attacks are forced

to be online, they are easier to detect and throttle, so the query-resource terms in our analytical results are likely capped at smaller values than when some or all of an attack can progress offline.

Our attacks exploit structural commonalities of CMS and HK. At their core, each of these processes incoming data elements by mapping them to multiple positions in an array of counters, and these are updated according to simple, structure-specific rules. Similarly, when frequency estimation (or *point*) queries are made, the queried element is mapped to its associated positions, and the response is computed as a simple function of values they hold. So, our attacks concern themselves with finding *cover sets*: given a target x, find a small set of data elements (not including x) that collectively hash to all the positions associated with x. Intuitively, inserting a cover set for x into the stream will give the structure incorrect information about x's relationship to the stream, causing it to over- or underestimate its frequency.

The existence of a cover set in the represented data is necessary for producing frequency estimation errors in HK, and both necessary and sufficient in CMS. Sadly, our findings suggest that preventing an adaptive adversary from finding such a set seems futile, no matter what target element is selected. The task can be made harder by increasing the structural parameters, but this quickly leads to structures whose size makes them unattractive in practice, i.e., *linear* in the length of the stream.

Motivating a more robust CFE Say that the array M in CMS has k rows and m counters (columns) per row. The CMS estimate for x is $\hat{n}_x = \min_{i\in[k]}\{M[i][p_i]\}$, where p_i is the position in row i to which x hashes. In the insertion-only stream model it must be that $\hat{n}_x \geq n_x$, where n_x is the true frequency of x. To see this, given an input stream $\vec{S}$, let $V_x^i = \{y \in \vec{S} \,|\, y \neq x \text{ and } h_i(y) = p_i\}$ be the set of elements that hash to the same counter as x, in the i-th row. Then we can write $M[i][p_i] = n_x + \sum_{y\in V_x^i} n_y$, where the $n_y > 0$ are the true frequencies of the colliding ys. Viewed this way, we see that the CMS estimate $\hat{n}_x$ minimizes the impact of "collision noise", i.e., $\hat{n}_x = n_x + \min_{i\in[k]}\{\sum_{y\in V_x^i} n_y\}$.

We could improve this estimate if we knew some extra information about the value of the sum, or the elements that contribute to it.

Let's say that, with a reasonable amount of extra space, we could compute $C_i = \epsilon_i \left(\sum_{y\in V_x^i} n_y\right)$ for some $\epsilon_i \in [0,1]$ that is bounded away from zero. Then we would improve the estimate to $\hat{n}_x = n_x + \min_{i\in[k]} \left\{(1-\epsilon_i)\left(\sum_{y\in V_x^i} n_y\right)\right\}$. How might we do this? Consider the case that for some row $i \in [k]$ there is an element $y^* \in V_x^i$ that dominates the collision noise, e.g. $n_{y^*} = (1/2)\sum_{y\in V_x^i} n_y$. Then even the ability to accurately estimate n_{y^*} would give a significant improvement in accuracy of $\hat{n}_x$, by setting C_i to this estimate. It turns out that HK provides something like this. It maintains a $k \times m$ matrix A, where $A[i][j]$ holds a pair (fp, cnt). In the first position is a *fingerprint* of the current "owner" of this position, and, informally, cnt is the number of times that $A[i][j]$ "remembers" seeing the current owner. (Ownership can change over time, as we describe in the body.) If we use the same hash functions to map element x into the same-sized M and A, then there is possibility of using the information at $A[i][p_i]$ to reduce the additive error (w.r.t. n_x) in the value of $M[i][p_i]$. This observation forms the kernel of our new Count-Keeper structure.

The Count-Keeper CFE We propose a new structure that, roughly speaking, combines equally sized (still compact) CMS and HK structures, and provide analytical and empirical evidence that it reduces the error (by at least a factor of two) that can be induced once a cover set is found. It also requires a type of cover set that is roughly twice as expensive (in terms of oracle queries)

to find. Moreover, it can effectively detect when the reported frequency of an element is likely to have large error. In this way we can dampen the effect of the attacks, by catching and raising a *flag* when a cover set has been found and is inserted many times to induce a large frequency error estimation on a particular element.

Intuitively, our Count-Keeper (CK) structure has improved robustness against adaptive attacks because CMS can only overestimate the frequency of an element, and HK can only underestimate the frequency (under a certain, practically reasonable assumption). We experimentally demonstrate that CK is robust against a number of attacks we give against the other structures. Moreover, it performs comparably well if not better than the other structures we consider in frequency estimation tasks in the non-adversarial setting.

As a side note, we uncovered numerous analytical errors in [55] that invalidate some of their claims about the behaviors of the HK structure. We have communicated with the authors of [55] and contacted Redis, whose RedisBloom library implements HK (and CMS) with fixed, public hash functions (i.e., the internal randomness is fixed for all time and visible to attackers).

3.2 Probabilistic Data Structures in the Wild: A Security Analysis of Redis

The following is a summary of our work that is currently available on ePrint [17]. It will soon be in submission at an appropriate venue.

Probabilistic data structures (PDS) are becoming ubiquitous in modern computing applications that deal with large amounts of data, especially when the data is presented as a stream. Their key property in this setting is that they provide approximate answers to queries on data without needing to store all the data. For example, a user may wish to estimate the cardinality of a datastream (in which case the HyperLogLog cardinality estimator could be used), find the most frequent elements in the stream (in which case a so-called top-K PDS is available), or just ask whether a particular data item has been seen before in the stream (where Bloom and Cuckoo filters are the tool for the job). Many modern data warehousing and processing systems provide access to PDS as part of their functionality.

A prominent example of such a system is Redis, a general purpose, in-memory database. Redis is integrated into general data analytics and computing platforms offered by AWS, Google Cloud, IBM Cloud, and Microsoft Azure, amongst others. Redis supports a variety of PDS: HyperLogLog (HLL), Bloom filter, Cuckoo filter, t-digest, Top-K, and Count-Min sketch [58]. While Redis was mostly used as a cache in the past, it is now a fully general system, used by companies like Adobe [59], Microsoft [60], Facebook [61] and Verizon [62] for a variety of purposes. These include security-related applications, such as traffic analysis and intrusion detection systems [63].

As the functionality of Redis has broadened, so has its maturity with respect to security. Initially, the Redis developers stated that no security should be expected from Redis: *The Redis security model is: "it's totally insecure to let untrusted clients access the system, please protect it from the outside world yourself"* [64]. In reality, users failed to comply with this [65]. Today, Redis has a number of security features, and has adopted a different model, with a protected mode as default, user authentication, use of TLS, and command blocklisting amongst other features [66]. Redis now also recognize security and performance in the face of adversarially-chosen inputs as being a valid concern, stating that *"an attacker might insert data into Redis that triggers pathological (worst case) algorithm complexity on data structures implemented inside Redis internals"* and then going on to discuss two potential issues, namely hash table exhaustion and worst-case sorting

behavior triggered by crafted inputs [66]. The first issue is prevented in Redis by using hash function seeding; the second issue is not currently addressed. However, Redis' consideration of malicious inputs does not seem to extend to their PDS implementations.

Given its prominence in the marketplace and the many other systems that rely on it, we contend that the PDS used in Redis are deserving of detailed analysis. Moreover, in view of the broad set of use cases for these PDS, including those where adversarial interference is anticipated and would be damaging if successful, this analysis should be done in an adversarial setting. This approach follows a line of recent work on PDS analysis [67, 68, 69, 70, 71, 72]. In this paper, we make a comprehensive security analysis of the suite of PDS provided by Redis, with a view to understanding how its constituent PDS perform in adversarial settings. As argued in [73], we regard the observation, documentation, and analysis of such security phenomena "in the wild" as constituting scientific contributions in their own right.

Following prior work, we assume only that the adversary has access to the functionality provided by the PDS (eg. via the presented API). The adversary's aim is then to subvert the main goal of the specific PDS under study. We deliberately remain agnostic about precisely which application is running on top of Redis, since the relevant applications will change over time and are anyway largely proprietary. The real-world effects of a successful attack will vary across applications, but might include, for example, false statistical information being presented to users (in the case of frequency estimation), wrongly reporting the presence of certain data items in a cache (in the case of Bloom filters or Cuckoo filters) leading to performance degradation, or the evasion of network attack detection (in the case of cardinality estimation being used in network applications). Instead of making application-specific analyses, we focus on the core PDS functionalities in Redis and how their goals can be subverted in general. Naturally, our analyses are specific to each of the different PDS supported in Redis, and depend on various low-level implementation choices made by Redis. These choices lead us to develop novel attacks that are more powerful than the known generic attacks against the different PDS in Redis.

Since HLL in Redis was already comprehensively studied in [14], we do not consider it further here. We note only that [14] showed how to manipulate data input to Redis HLL to distort cardinality estimates in severe ways, in a variety of adversarial settings.

The t-digest is a data structure first introduced in [74]; it uses a k-means clustering technique [75] to estimate percentiles over a collection of measurements. The structure is an outlier in the Redis PDS suite as it does not work in the streaming setting, but necessitates the batching of data in memory, and it is not really probabilistic in the same sense as the other PDS in Redis. For these reasons, we omit a security evaluation of t-digest (both in general and in the case of the Redis implementation).

This leads us to focus on the remaining four PDS in Redis: Bloom filter, Cuckoo filter, Top-K, and Count-Min sketch. For each PDS, we discuss how the PDS was originally described in the literature and lay out how the Redis implementation differs from this "theoretical" description. We then develop attacks for each of these four PDS, with the attacks in most cases exploiting specific features of the Redis implementations and being more efficient for this reason (simultaneously, we have to deal with the many oddities of the Redis codebase in our attacks). In total, we present 10 different attacks across the four PDS. We compare our attacks with known attacks for these PDS from the literature. We also look at how the PDS in Redis can be protected against attacks, drawing on existing literature that considers this question for PDS more generally [11, 13, 14, 72].

We give a brief flavour of our attacks on the Redis PDS suite. For the Bloom filter implemen-

tation, we show how to make any target data item a false positive with few insertions. For the Cuckoo filter, we show how to launch an attack that disables insertions after only a few insertions have been made, far fewer than the filter's expected capacity. For Count-Min sketch, we can inflate the frequency estimate of any target data item to any target level. For Top-K, we can block the PDS from reporting the true K most frequent data items in a stream.

4 Proposed Work

4.1 Skipping Data Structures in Adversarial Environments

Recall the attacks that involve exploiting weak choices of hash functions to force worst-case run time behavior in hash tables, and the attacks that exploit knowledge of the random choice of height of an element in a skip list to degenerate the structure. While many papers have explored various attacks and proposed a number of possible mitigations, we are the first to formalize a security model for this class of structure and provide efficient and secure constructions for the hash table and skip list in this model.

Our goal is to capture the average-case run time of operations skipping data structures being conserved in the face of an adaptive adversary that can control the data being represented by the structure. Loosely, the average-case run time of skipping data structures is related to how data is "distributed" in the representation. For instance, an ideal hash table would distribute the elements it represents equally among the buckets. Analogously, ideal ordered structures (e.g., a skip list or a treap) would resemble a balanced tree. If a data collection was fixed, and we ignored a desire for efficiency, one could always craft an ideal representation with respect to the runtime of queries. For a hash table, one could find a hash function that equally distributes the fixed collection to its buckets. For a fixed-ordered structure, one could simply assign the heights (depths) of elements such that the shortest possible search paths are guaranteed, as in a perfectly balanced tree structure.

However, skipping structures are used in a mutable setting. For this reason (and for efficiency), skipping data structures use some form of randomness to process updates dynamically and in turn update their representation. Hash tables select a random hash function to map elements to buckets, and (generally) the ordered skipping structures flip coins at insertion time to determine the height of an element. These processes have been shown (with high probability) to yield representations of a dynamic data collection that are "close" to the ideal representations. Hash tables are analyzed using standard ball-and-bin arguments. Assuming a collision-resistant hash function and a load factor such that $n \approx b$ (i.e., the size n of the data collection stored is about equal to the number b of buckets), it is known [76] that with probability $p = 1 - \frac{1}{b}$ that at any point in time no bucket has more than $3\frac{\log b}{\log \log b}$. This maximum bucket population bounds directly corresponds with the maximum insertion, deletion, or query time of a subsequent operation. Likewise, the search cost path of any element queried to a skip list has been shown with high probability to not exceed $O(\log n)$ (where the exact constants are functions of the parameters of the structure).

These analyses are done under a strictly *non-adaptive* adversarial assumption. That is, these probabilistic bounds on the "distribution" of elements are done under the assumption that the updates and queries made to the structure do not depend on the internal randomness of the structure, the results of past operations, or the state of the representation. In the adaptive adversarial setting, this cannot be assumed. This is seen in both the hash flooding attack and the skip list degeneration attack. Therefore, intuitively, a robust skipping data structure would conserve the desired element distribution property of the structure with high probability, even in the face of an

adaptive adversary. This is what we aim to capture with our novel formal security model.

Further, we explore how to make these structures adversarial robust as efficiently as possible. We observe that two abilities allow an adaptive adversary to shape the distribution of data in a skipping structure such that subsequent operations on the structures are worst-case time with high probability. The first is the ability of the adversary to influence *how or where* a particular element gets placed in the structure upon insertion. This is akin to being able to know a priori what bucket an element will be inserted to in a hash table or what height an element will be inserted at in a skip list. The second is the ability to *delete* elements. This provides a mechanism for an adversary to degenerate a structure after a series of insertions by deleting unfavorable (w.r.t. to the goal of the adversary) elements.

Therefore, we propose two inexpensive modifications to the base skipping structures. We will later prove these modified structures secure in our security model. The first is to swap hash functions for secretly keyed primitives (concretely a PRF) for both hash tables and the deterministic versions of the other skipping structures. This serves to prevent trivial hash flood style attacks. The second is to prevent the adversary from actually deleting elements from the structure. This stultifies the ability of an adversary to perform a skip list degeneration style attack, even in the case it has full access to the internal state of the data structure.

Removing the deletion functionality entirely from our data structure would be undesirable. Instead, we use a simple scheme that allows for the removal of elements without modifying the actual underlying structure of a skipping DS that has been imposed by previous insertions. We achieve this by replacing the label of the element (e.g., the key-value data) with a distinguished symbol $\diamond$, but not modifying (say) the linked list in a hash table bucket or the skip list structure by removing the node (and its associated height) where this element previously existed. For the hash table. on subsequent insertions, if an element can be placed in the empty node of a previously deleted element, we do this first before allocating a new node via the usual insertion process. For a skip list, this is not possible due to a subtle attack that exists if we tried this replacement strategy. Instead, we must perform periodic cleanups of the deleted elements.

This change prevents the adversary from eliminating desired skip connections in a skip list or obtaining trivial wins in our security model against a hash table (when taking the represented set to be the collection of all empty and non-empty elements). However, this modified deletion functionality affects the space efficiency of the structures and the efficiency of the range query functionality for the ordered skipping structures. In this work, we will not only prove our constructions secure, but we also discuss approaches to ameliorating such concerns and analyze the trade-offs of these approaches.

4.2 Solving Scalability for Key Transparency Systems via a Privacy-Preserving Verifiable Summary Mechanism

Key Transparency (KT) systems offer a solution to the challenges of public key distribution in end-to-end encrypted communication platforms. KT systems are vital to preventing trivial man-in-the-middle attacks. Traditionally, verifying the authenticity of another party's public key in secure communications required either physical meetings to exchange keys—a cumbersome process, especially with frequent key rotations and new device additions—or reliance on a third-party authority. KT systems address these challenges by providing an automated mechanism. This mechanism allows users to verify that they are receiving the correct public key, or at least one that is consistent with what other users are seeing from the same service. At the same time, it preserves

user privacy. While KT systems cannot prevent misbehavior by a service provider, they ensure that such misbehavior is quickly detectable. KT systems have not only attracted significant academic interest, as evidenced by studies such as [77, 78, 49, 79, 80, 81, 82, 83, 84, 85, 50], but have also been realized by platforms such as Keybase [86], Zoom [87], Google [88], WhatsApp [89], Apple iMessage [90], and Proton [91].

Scalability is the primary challenge in real-world KT system deployments, particularly for large services with billions of users and multiple keys per user. Academic work has often overlooked the performance and architectural complexities of such large-scale systems. For instance, WhatsApp's KT deployment features a key map with 50 billion nodes and processes 150,000 updates every five minutes, demanding vast storage and computational resources [92]. Even with more efficient protocols like Parakeet [84] and OPTIKS [50], storing such a large key log in RAM is impractical, necessitating external storage layers that can slow query responses. Additionally, distributed KT systems face significant challenges in maintaining consistency across geographically disparate nodes, as any inconsistency in updates can compromise the system's guarantees. While WhatsApp mitigates this by using a single write node, this approach introduces a potential bottleneck and a single point of failure. Moreover, users expect seamless performance, so it's crucial to develop scalable KT solutions that minimize overhead and provide quick, low-bandwidth responses. To address these challenges, we propose a new scalable KT system leveraging a verifiable summary mechanism (using verifiable bloom filters) that reduces the need for full KT key log access.

In particular, this work makes the following contributions:

- We present generic and extensible definitions for KT systems. Previous works [77, 49, 84] have treated key transparency systems as a monolithic block, i.e., they blend together the specification of a KT system, and the protocol that realizes that specification. This significantly complicates the analysis of the protocols and the interpretation of the derived security guarantees.

  We notice that this was also the case in the early stages of the development of other multi-party computation (MPC) applications, like key exchange [93]. However, nowadays, there is a clear distinction between a public-key encryption (PKE) *scheme*[1] and a key-exchange protocol *protocol*.

  In the context of MPC, the objective of an MPC is captured in the form of an *ideal* functionality that specifies, how the system should behave in an idealized setting. A protocol for that functionality is then judged to be secure, if it realizes the functionality in the real-ideal simulation paradigm[94].

  Since KT is inherently an MPC, we deem it appropriate and important to formally specify an ideal functionality for KT. We note that [49] initiate the formalization of key transparency though the notion of a verifiable key directory (VKD). Though, while formally defining several algorithms, their definition lacks the previously mentioned distinction between a scheme and a protocol. We argue that in order to properly formalize key transparency, we need to distinguish between three formal concepts:

  1. a KT *functionality* that specifies what the formal objective of a key transparency system is (e.g. supplying users with previously registered keys),

[1] A formal PKE scheme does not impose a notion of a sender or receiver, only encryption and decryption (and key generation).

2. a KT *protocol* that realizes the KT functionality (using a KT scheme), and

3. a KT *scheme* that defines a set of algorithms (analogous to a PKE scheme).

- We present a novel primitive – the verifiable bloom filter verifiable summary. This mechanism allows one to answer set membership queries in a verifiable and privacy preserving manner while only requiring a very small amount of space (relative to the set that is represented). It consists of a composition of a VRF [95] and a standard Bloom filter.
- Furthermore, we detail our full scalable KT system that uses this verifiable summary mechanism. We also show how our verifiable summary mechanism is largely separable from existing KT solutions, making it adaptable to many current (and presumably future) KT systems that natively lack this scalability feature.
- We implement the core components of our system using the AKD library from Facebook [96]. Further, we implement a simulator to analyze our full protocol. We find that our system greatly reduces queries to the main KT server and provides a significant speed-up in query response time to clients.

4.3 Compact, Private, and Verifiable Data Structures

Recall that a Bloom filter [2] is the canonical example of a *probabilistic data structure*: it is a constant-sized array of m bits that summarizes a collection of $N \gg m$ data elements, in a manner that admits efficient responses to the specific query type: *Is data element x present in the collection?* Because of the significant compression, the *yes/no* responses to these queries are only guaranteed to be correct with some probability, this probability being over certain random choices that are made prior to the creation of the summary from the data. Implicit in the probabilistic guarantee is that the data is independent of the Bloom filter's random choices, in the sense that the data can be treated as having been fixed before those choices were made. Due to their increasingly widespread use in systems that are deployed in untrusted environments, where the data cannot be assumed to be independent of the filter's randomness (e.g., when the data is streaming, and interleaved with queries), recent work has revisited the classical correctness guarantees in the context of adversarially shaped data. A number of recent works have examined the correctness guarantees of these structures in adversarial settings [10, 11, 14, 13, 72], as well as initiating explorations of privacy for the represented data [97]. The latter is important for applications, some of which we will target in this work, where the data itself is sensitive, either directly or by what they imply.

These recent results assume an attack model in which the threat is external, in the sense that the summary bit-array is assumed to have been produced correctly from *whatever* data is provided, and query responses are assumed to be correctly computed, no matter how the queries are crafted. For measuring the accuracy of responses (i.e., data structure correctness) under adversarial data and queries, or for arguing about privacy of the represented data, this attack model makes intuitive sense. But what about applications in which the querying party is honest, and does *not* necessarily trust that the summary computation or query responses have been honestly executed?

Such applications are the focus of this work. We aim to build data structures with query responses that are *publicly verifiable* as having been correctly computed and consistent with what querying parties know about the data (and its representation), while guaranteeing minimal leakage of information about the data (i.e., beyond what would be leaked as a result of being operationally correct and useful), and while imparting as little space and efficiency overhead as possible, with

respect to classical structures that might be used if security concerns were removed.

For instance, recall the verifiable summary mechanism from Section 4.2. This can be seen as a *verifiable Bloom filter* (VBF), constructed via simple composition of a verifiable random function (VRF) [95] and a standard Bloom filter.

Realizing Query Verifiable Data Structures The VBF is one concrete instantiation of a general approach to realizing our central primitive, the *query verifiable data structure* (QVDS). Informally, a QVDS is an immutable data structure (in the syntax of [11]) that has been embellished to support verification of its responses to queries. We name our general approach to constructing QVDS the "VRF-then-DS" composition, because it is highly suggestive of what it is: first, apply a VRF to the data, and then treat the corresponding VRF outputs as the "data" collection to be represented by a traditional DS that fits the desired query type(s). For example, for applications where verifiable element-frequency queries are needed, the VRF-then-DS composition could be applied to a count-min sketch [4].

After establishing the syntax for our QVDS abstraction, we set out notions of completeness, verifiability, correctness and privacy. Loosely speaking, completeness demands that honestly generated query responses must verify; verifiability requires that dishonestly produced responses will not verify. Because traditional data structures offer a variety of correctness properties, our notion of QVDS correctness is parameterized by a correctness relation, which may itself be a set-theoretic function of multiple relations. For example, correct Bloom filters have the property of there being no false negatives when responding to queries, and they have a bounded false-positive property, where the bound is a function of the data collection size, and the Bloom filter parameters m, k. Each of these can be captured as a stand-alone correctness relation. Similarly, our notion of QVDS privacy is parameterized by a privacy relation that captures what should be considered "secret". For example, if the data objects are pairs $(\mathsf{id}_i, \mathsf{balance}_i)$, one could demand that no adversary can determine any given $\mathsf{balance}$ that is not theirs (allow for insider attacks); or, that the no adversary can link together a particular $\mathsf{balance}$ and user ID id. By parameterizing our correctness and privacy notions, we are able to capture properties for disparate QVDS (especially when building from existing, traditional DS), and this approach may be of independent interest.

We observe that common types of queries, including membership queries and element-frequency queries, can be responded to without knowledge of the *particular* data elements that captured in the DS representation. One could arbitrarily relabel all the points in the universe of data elements and, so long as the labels were unique and fixed, there would be no change to the query responses. In effect, the data structure would treat the collection of labels as the data. When answers to (remapped) queries are invariant to the relabeling of the data collection S to some data-agnostic collection Z, we show that average-case correctness guarantees[2] suffice for a VRF-then-DS construction to achieve our (worst-case) notion of QVDS correctness. Relaxing the DS-correctness requirements, which are often defined for the worst-case, may allow VRF-then-DS constructions to use underlying data structures that, while more efficient in terms of time and space complexity, would otherwise be discarded for lack of usefulness on their own.

[2]Loosely, for any fixed collection of labels, the DS would be deemed "correct" if, its traditional correctness properties held for a random collection of *actual* data that would result in those labels.

5 Timeline

The following is an outline of work that I have already completed towards my thesis (and some that falls outside my thesis), as well as a timeline of proposed works. For this reason, this section is not written with the usual academic formalisms – namely the use of the pronouns *I* and *my*.

- *Compact Frequency Estimators in Adversarial Environments* [16] was presented at the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS 2023).
- *Probabilistic Data Structures in the Wild: A Security Analysis of Redis* [17] is in complete and will be in submission at an appropriate venue soon.
- The proposed work on *Solving Scalability for Key Transparency Systems via a Privacy-Preserving Verifiable Summary Mechanism* is underway and will be submitted to the 46th IEEE Symposium on Security and Privacy on November 14, 2024. The follow-on work, *Compact, Private, and Verifiable Data Structures*, is also underway and will be submitted to an appropriate conference or journal in Spring 2025.
- The proposed work on *Skipping Data Structures in Adversarial Environments* is underway and will be submitted to an appropriate conference or journal in Spring 2025. I am traveling to Technische Universität Darmstadt in the latter half of November 2024 to work with my collaborators on this project.
- After the completions of these proposed works and their subsequent submissions, I will spend Summer and (the first half of) Fall 2025 writing my thesis[3]. I plan on defending in the latter half of the Fall of 2025.

I would also like to mention two publications that I have contributed during my graduate work that will not be included as part of my thesis.

- *The DecCert PKI: A Solution to Decentralized Identity Attestation and Zooko's Triangle* [98] was presented at the 2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS) and received the best paper award.
- *Leveraging Generative Models for Covert Messaging: Challenges and Tradeoffs for "Dead-Drop" Deployments*[99] was presented at the Fourteenth ACM Conference on Data and Application Security and Privacy (CODASPY '24).

References

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, third edition edition, 2009.

[2] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science*, (Proceedings), 2007.

[3] In the event of any rejections, this time will also be used to revise and resubmit these works.

[4] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[5] Roberto Tamassia. Authenticated data structures. In *Algorithms-ESA 2003: 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003. Proceedings 11*, pages 2–5. Springer, 2003.

[6] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):1–26, 2008.

[7] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.

[8] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys & Tutorials*, 14(1):131–155, 2011.

[9] Bob Goodwin, Michael Hopcroft, Dan Luu, Alex Clemmer, Mihaela Curmei, Sameh Elnikety, and Yuxiong He. Bitfunnel: Revisiting signatures for search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 605–614, 2017.

[10] Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In *Annual Cryptology Conference*, pages 565–584. Springer, 2015.

[11] David Clayton, Christopher Patton, and Thomas Shrimpton. Probabilistic data structures in adversarial environments. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1317–1334, 2019.

[12] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM transactions on networking*, 8(3):281–293, 2000.

[13] Mia Filic, Kenneth G Paterson, Anupama Unnikrishnan, and Fernando Virdia. Adversarial correctness and privacy for probabilistic data structures. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1037–1050, 2022.

[14] Kenneth G Paterson and Mathilde Raynal. Hyperloglog: Exponentially bad in adversarial settings. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 154–170. IEEE, 2022.

[15] Pedro Reviriego and Daniel Ting. Security of hyperloglog (hll) cardinality estimation: Vulnerabilities and protection. *IEEE Communications Letters*, 24(5):976–980, 2020.

[16] Sam A Markelon, Mia Filic, and Thomas Shrimpton. Compact frequency estimators in adversarial environments. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 3254–3268, 2023.

[17] Mia Filić, Jonas Hofmann, Sam A Markelon, Kenneth G Paterson, and Anupama Unnikrishnan. Probabilistic data structures in the wild: A security analysis of redis. *Cryptology ePrint Archive*, 2024.

[18] Scott A Crosby and Dan S Wallach. Denial of service via algorithmic complexity attacks. In *12th USENIX Security Symposium (USENIX Security 03)*, 2003.

[19] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[20] Alexander Klink and Julian Walde. Efficient denial of service attacks on web application platforms. In *28th Chaos Communication Congress*, 2011.

[21] Jean-Philippe Aumasson, DJ Bernstein, and M Boblet. Hash-flooding dos reloaded: attacks and defenses. In *29th Chaos Communications Congress*, 2012.

[22] Austin Appleby. Smhasher. `https://github.com/aappleby/smhasher`, 2016.

[23] Jean-Philippe Aumasson and Daniel J Bernstein. Siphash: a fast short-input prf. In *International Conference on Cryptology in India*, pages 489–508. Springer, 2012.

[24] Jean-Philippe Aumasson, Martin Boßlet, and Daniel J Bernstein. Hash-flooding dos reloaded:attacks and defenses. Slides presented at the 2011 Application Security Forum – Western Switzerland, October 2011.

[25] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[26] Eyal Nussbaum and Michael Segal. Skiplist timing attack vulnerability. In *International Workshop on Data Privacy Management*, pages 49–58. Springer, 2019.

[27] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. *IEEE Journal on selected areas in communications*, 18(4):561–570, 2000.

[28] Paul C Kocher. On certificate revocation and validation. In *International conference on financial cryptography*, pages 172–177. Springer, 1998.

[29] Irene Gassko, Peter S Gemmell, and Philip MacKenzie. Efficient and fresh certification. In *Public Key Cryptography: Third International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2000, Melbourne, Victoria, Australia, January 18-20, 2000. Proceedings 3*, pages 342–353. Springer, 2000.

[30] Ahto Buldas, Peeter Laud, and Helger Lipmaa. Accountable certificate management using undeniable attestations. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 9–17, 2000.

[31] Michael Shin, Christian Straub, Roberto Tamassia, and Daniel J Polivy. Authenticating web content with prooflets. *Technical report, Center for Geometric Computing, Brown University*, 2002.

[32] Daniel J Polivy and Roberto Tamassia. Authenticating distributed data using web services and xml signatures. In *Proceedings of the 2002 ACM workshop on XML security*, pages 80–89, 2002.

[33] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[34] Ralph C Merkle. Method of providing digital signatures, January 5 1982. US Patent 4,309,569.

[35] Aris Anagnostopoulos, Michael T Goodrich, and Roberto Tamassia. Persistent authenticated dictionaries and their applications. In *International Conference on Information Security*, pages 379–393. Springer, 2001.

[36] 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.

[37] Michael T Goodrich, Roberto Tamassia, Nikos Triandopoulos, and Robert Cohen. Authenticated data structures for graph and geometric searching. In *Cryptographers' Track at the RSA Conference*, pages 295–313. Springer, 2003.

[38] Charles Martel, Glen Nuckolls, Premkumar Devanbu, Michael Gertz, April Kwong, and Stuart G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, jan 2004.

[39] Michael T Goodrich, Roberto Tamassia, and Jasminka Hasić. An efficient dynamic and distributed cryptographic accumulator. In *International Conference on Information Security*, pages 372–388. Springer, 2002.

[40] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Authenticated hash tables based on cryptographic accumulators. *Algorithmica*, 74:664–712, 2016.

[41] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 91–110, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[42] Andrew Miller, Michael Hicks, Jonathan Katz, and Elaine Shi. Authenticated data structures, generically. *ACM SIGPLAN Notices*, 49(1):411–423, 2014.

[43] Silvio Micali, Michael Rabin, and Joe Kilian. Zero-knowledge sets. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 80–91. IEEE, 2003.

[44] Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin. Mercurial commitments with applications to zero-knowledge sets. In *Advances in Cryptology–EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24*, pages 422–439. Springer, 2005.

[45] Rosario Gennaro and Silvio Micali. Independent zero-knowledge sets. In *International Colloquium on Automata, Languages, and Programming*, pages 34–45. Springer, 2006.

[46] Moses Liskov. Updatable zero-knowledge databases. In *Advances in Cryptology-ASIACRYPT 2005: 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005. Proceedings 11*, pages 174–198. Springer, 2005.

[47] Dario Catalano, Dario Fiore, and Mariagrazia Messina. Zero-knowledge sets with short proofs. In *Advances in Cryptology–EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings 27*, pages 433–450. Springer, 2008.

[48] Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In *Theory of Cryptography Conference*, pages 499–517. Springer, 2010.

[49] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. Seemless: Secure end-to-end encrypted messaging with less trust. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1639–1656, 2019.

[50] Julia Len, Melissa Chase, Esha Ghosh, Kim Laine, and Radames Cruz Moreno. Optiks: An optimized key transparency system. *Cryptology ePrint Archive*, 2023.

[51] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. Deriving traffic demands for operational ip networks: Methodology and experience. In *AMC SIGCOMM*, 2000.

[52] Anukool Lakhina, Mark Crovella, and Christiphe Diot. Characterization of network-wide anomalies in traffic flows. In *ACM SIGCOMM Conference on Internet Measurement*, 2004.

[53] Luca Melis, George Danezis, and Emiliano De Cristofaro. Efficient private statistics with succinct sketches. In *Proceedings of the NDSS 2016*, 2016.

[54] Redisbloom: Probabilistic data structures for redis. `https://oss.redis.com/redisbloom/`.

[55] Tong Yang, Haowei Zhang, Jinyang Li, Junzhi Gong, Steve Uhlig, Shigang Chen, and Xiaoming Li. Heavykeeper: an accurate algorithm for finding top-k elephant flows. *IEEE/ACM Transactions on Networking*, 27(5):1845–1858, 2019.

[56] Benedikt Sigurleifsson, Aravindan Anbarasu, and Karl Kangur. The count-min sketch data structure and its uses within computer science, 2019.

[57] Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. `https://redis.io/`.

[58] Probabilistic: Probabilistic data structures in Redis. `https://redis.io/docs/data-types/probabilistic/`.

[59] Robert Taylor RedisConf 2021. How Adobe uses the Enterprise tier of Azure Cache for Redis to serve push notifications, Adobe, 2024. `https://www.youtube.com/watch?v=OslaeJEXW5k`.

[60] Charles Morris RedisDays New York 2022. Using AI to Reveal Trading Signals Buried in Corporate Filings, 2024. `https://www.youtube.com/watch?v=_Lrbesg4DhY`.

[61] Guy Yonish Redis Day TLV 2016. Redis @ Facebook, 2024. `https://www.youtube.com/watch?v=XGxntWcjI24`.

[62] Robert Belson RedisConf 2021. Redis on the 5G Edge: Practical advice for mobile edge computing, Verizon, 2024. `https://www.youtube.com/watch?v=NwQwE2JAIXc`.

[63] Kurt John The Data Economy Podcast. Using Real-Time Data and Digital Twins to Improve Cyber Security, 2024. `hhttps://www.youtube.com/watch?v=TycylT0J6cc`.

[64] Salvatore Sanfilippo. A few things about redis security. `http://antirez.com/news/96`.

[65] Tobias Fiebig, Anja Feldmann, and Matthias Petschick. A one-year perspective on exposed in-memory key-value stores. In Nicholas J. Multari, Anoop Singhal, and David O. Manz, editors, *Proceedings of the 2016 ACM Workshop on Automated Decision Making for Active Cyber Defense, SafeConfig@CCS 2016, Vienna, Austria, October 24, 2016*, pages 17–22. ACM, 2016.

[66] Redis security: Security model and features in Redis. `https://redis.io/docs/latest/operate/oss_and_stack/management/security/`.

[67] Thomas Gerbet, Amrit Kumar, and Cédric Lauradoux. The power of evil choices in bloom filters. In *DSN*, 2015.

[68] David Clayton, Christopher Patton, and Thomas Shrimpton. Probabilistic data structures in adversarial environments. In *ACM SIGSAC CCS*, 2019.

[69] Damien Desfontaines, Andreas Lochbihler, and David Basin. Cardinality Estimators do not Preserve Privacy. In *Privacy Enhancing Technologies*, pages 26–46, 2019.

[70] Pedro Reviriego and Daniel Ting. Security of hyperloglog (HLL) cardinality estimation: Vulnerabilities and protection. *IEEE Commun. Lett.*, 24(5):976–980, 2020.

[71] Kenneth G. Paterson and Mathilde Raynal. Hyperloglog: Exponentially bad in adversarial settings. In *EuroS&P*, 2022.

[72] Sam A. Markelon, Mia Filić, and Thomas Shrimpton. Compact frequency estimators in adversarial environments. In *ACM SIGSAC CCS*, 2023.

[73] Martin R. Albrecht and Kenneth G. Paterson. Analysing cryptography in the wild - a retrospective. Cryptology ePrint Archive, Paper 2024/532, 2024. `https://eprint.iacr.org/2024/532`.

[74] Ted Dunning. The t-digest: Efficient estimates of distributions. *Software Impacts*, 7:100049, 2021.

[75] Trupti M Kodinariya, Prashant R Makwana, et al. Review on determining number of cluster in k-means clustering. *International Journal*, 1(6):90–95, 2013.

[76] Shuchi Chawla. Cs787: Advanced algorithms scribe notes, lecture 7: Randomized load balancing and hashing, September 2009.

[77] 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.

[78] Joseph Bonneau. Ethiks: Using ethereum to audit a coniks key transparency log. In *International Conference on Financial Cryptography and Data Security*, pages 95–105. Springer, 2016.

[79] 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.

[80] Ioanna Tzialla, Abhiram Kothapalli, Bryan Parno, and Srinath Setty. Transparency dictionaries with succinct proofs of correct operation. *Cryptology ePrint Archive*, 2021.

[81] Yuncong Hu, Kian Hooshmand, Harika Kalidhindi, Seung Jin Yang, and Raluca Ada Popa. Merkle 2: A low-latency transparency log system. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 285–303. IEEE, 2021.

[82] Nirvan Tyagi, Ben Fisch, Andrew Zitek, Joseph Bonneau, and Stefano Tessaro. Versa: Verifiable registries with efficient client audits from rsa authenticated dictionaries. In *Proceedings*

of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pages 2793–2807, 2022.

[83] Brian Chen, Yevgeniy Dodis, Esha Ghosh, Eli Goldin, Balachandar Kesavan, Antonio Marcedone, and Merry Ember Mou. Rotatable zero knowledge sets: Post compromise secure auditable dictionaries with application to key transparency. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 547–580. Springer, 2022.

[84] Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Oztürk, Kevin Lewi, and Sean Lawlor. Parakeet: Practical key transparency for end-to-end encrypted messaging. *Cryptology ePrint Archive*, 2023.

[85] Julia Len, Melissa Chase, Esha Ghosh, Daniel Jost, Balachandar Kesavan, and Antonio Marcedone. Elektra: Efficient lightweight multi-device key transparency. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2915–2929, 2023.

[86] Antonio Marcedone. Key transparency at keybase and zoom. Presentation at IETF 116 Meeting, March 2023. `https://datatracker.ietf.org/meeting/116/materials/slides-116-keytrans-keybase-and-zoom-00.pdf`.

[87] Josh Blum, Simon Booth, Brian Chen, Oded Gal, Maxwell Krohn, Julia Len, Karan Lyons, Antonio Marcedone, Mike Maxim, Merry Ember Mou, et al. Zoom cryptography whitepaper. 2022.

[88] Ryan Hurst and Gary Belvin. google/keytransparency. `https://github.com/google/keytransparency/`, 2020.

[89] Kevin Lewi. Whatsapp key transparency. In *Proceedings of the 2023 USENIX Conference on Privacy Engineering Practice and Respect (PEPR '23)*, Santa Clara, CA, 2023.

[90] Apple Security Engineering and Architecture (SEAR). imessage contact key verification. `https://security.apple.com/blog/imessage-contact-key-verification`, 2023. Accessed: 2023-04-01.

[91] Thore Göbel and Daniel Huigens. Proton key transparency whitepaper. 2024.

[92] Sean Lawlor and Kevin Lewi. Whatsapp key transparency, 2024. Real World Cryptography 2024.

[93] Whitfield Diffie and Martin E Hellman. Multiuser cryptographic techniques. In *Proceedings of the June 7-10, 1976, national computer conference and exposition*, pages 109–112, 1976.

[94] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13:143–202, 2000.

[95] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[96] Sean Lawlor and Kevin Lewi. Akd. `https://github.com/facebook/akd`, 2024.

[97] Ryo Nojima and Youki Kadobayashi. Cryptographically secure bloom-filters. *Trans. Data Priv.*, 2(2):131–139, 2009.

[98] Sam A Markelon and John True. The deccert pki: A solution to decentralized identity attestation and zooko's triangle. In *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 74–82. IEEE, 2022.

[99] Luke A Bauer, James K Howes, Sam A Markelon, Vincent Bindschaedler, and Thomas Shrimpton. Leveraging generative models for covert messaging: Challenges and tradeoffs for" deaddrop" deployments. In *Proceedings of the Fourteenth ACM Conference on Data and Application Security and Privacy*, pages 67–78, 2024.