On the Security of Data Structures

Qualifying Exam

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

Abstract—This survey looks at the application of formal security analysis to three classes of data structures: compact probabilistic data structures, fast average-case runtime data structures that are vulnerable to complexity attacks, and verifiable data structures. Recent research highlights the need to consider the security of data structures when their properties (e.g., correctness and run-time) depend in some way on the data they represent, especially when an adversary is able to control this data. We review the key works concerning the adversarial correctness of probabilistic data structures and complexity attacks against fast average-case runtime data structures. Was also provide comprehensive case studies showcasing how formal security analysis of these structures is done. In contract to the other classes of structures we consider, verifiable data structures are designed with explicit security notions. Instead, we explore a mismatch between the needs of key transparency systems (which rely on verifiable structures) and the guarantees provided by the traditional verifiable structure literature, emphasizing the importance of careful consideration when crafting security definitions.

1 Introduction

Data structures define representations of possibly dynamic (multi)sets, along with 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 data structures do not behave as expected when in the presence of adversaries that have the ability to control the data they represent. These results illustrate the need to rigorously analyze certain classes of data structures using the provable security paradigm. In this paper, we give a thorough discussion on the key works from three classes of data structures: compact probabilistic data structures, fast average-case runtime data structures, and verifiable data structures. We briefly introduce each of these class of data structure below and provide an integrated roadmap for the rest of our paper.

Probabilistic data structures (PDS) 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. In Section 2 we will survey these works and describe their efforts to retain the correctness guarantees of PDS when this data independence assumption does not hold. We also provide a detailed case-study analyzing the count-min sketch using a provable security style treatment. We end by highlighting open questions and provide brief descriptions of potential directions for future research.

In Section 3 we turn our attention towards the class of data structures that are susceptible to so called "complexity attacks". In contrast to 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, offer fast average-case runtime of their operations, but have worst-case runtime that is poor (in terms of the requirements of real-world applications). 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 now focus on the expected run time of these structures in the presence of an adversary. We review prior work on complexity attacks against hash tables and skip lists [5]. Unfortunately, there is a lack of formalization of these attacks and an absence of provable security treatment of proposed countermeasures, so we give a case study exploring how one may go about securing hash tables against these attacks from a provable security perspective. Finally, we outline open challenges in securing data structures of this nature.

Lastly, we shift directions to discussing 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. Verifiable (or authenticated) data structures have existed in the literature for decades [6]. Traditionally, these works assume that 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[7], [8], [9], [10] and downloading content from (untrusted) internet mirrors [11], [12].

In Section 4, we first survey the prior work in the verifiable data structure space. We then review the work that has been done on a special class of verifiable data structures called *zero-knowledge sets*. In addition to the verifiability property referred to above, this zero-knowledge set family of structures have strong privacy properties. We finish the section with a case study discussing the use of verifiable structures in *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 key transparency systems, and the security properties that are captured in the traditional verifiable structure literature.

2 Probabilistic Data Structures

2.1 Streaming Setting

First, we note that a number of PDS are often used in an online setting where data elements are inserted and subsequently "forgotten". This notion is formalized by the *streaming setting*, where a *stream* of data $\vec{S} = e_1, e_2, \ldots$ is a finite sequence of elements $e_i \in \mathcal{U}$ for some universe $\mathcal{U}$. The elements of a stream can have > 1 frequency (that is, they need not be distinct), and we define the frequency of some $x \in \mathcal{U}$ as $\hat{n}_x = |\{i : e_i = x|\}$. In the streaming setting, the stream is presented element-by-element, with no buffering or "look-ahead". More specifically, the stream is processed in order, and the processing of e_i is completed before the processing of e_{i+1} begins and once e_i is processed it cannot be recalled. The streaming setting is extensively used in the count-min sketch case study in Subsection 2.5. Moreover, a precursor to the work on the provable security of probabilistic data structures focused on two-party sketching protocols in the presence of an adversarially generated stream [13]. In particular, the work focused on the ability of the two parties to jointly compute a function when their inputs (which are not known to the other party) are adaptively chosen by an adversary. The inputs are provided as a stream with a total size larger than the space available to any of two honest parties, so they must keep and store their inputs in a compressed representation. The authors constructed adversarially robust solutions to computing set equality and the approximate size of the symmetric difference in this setting.

2.2 Bloom Filters and Approximate Set Membership PDS

The first works to explore PDS in a provable security paradigm 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 [14], networking [15], distributed systems [16], and search [17]. All of these contexts have clear potential adversarial incentives to disrupt expected operation, but traditionally such threats are not considered – especially as it comes to the Bloom filter's correctness guarantees in such conditions.

A Bloom filter represents a set $\mathcal{S}$ of data as a length m bit-array where m is much smaller than the actual number of bits needed to store $\mathcal{S}$ in its entirety. Initially, all m bits are set to 0. An element $x \in \mathcal{S}$ is added to the Bloom filter by computing k hash values $h_1(x), h_2(x), \ldots, h_k(x) \in [m]$ and setting all the corresponding array positions to 1. A set membership query for any $y \in \mathcal{S}$ is answered by running y through the k hash functions as before and responding positively if and only if all positions corresponding to the output of the hash functions hold a 1 bit (and negatively otherwise). False negative responses are not possible, but false positives exist. We point the reader to [18] for an analysis of the false positive rate in the non-adversarial setting.

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 [19]. 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. Such attacks have been shown to be able to disrupt real-world systems [20]. In response, 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. [21] extend the work of Naor and Yogev 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. Further, they formalize a notion of adversarial correctness that extends further than only Bloom filters. They consider a large class of abstract probabilistic data structures, rather than just approximate set-membership data structures. In their work, they also concretely analyze the counting filter [22] and the Count-min sketch [4].

Specifically for the Bloom filter, Clayton et al. show that the basic structure is insecure, in that the false positive rate on any query can be made arbitrarily close to one. In the immutable setting (where an adversary is not allowed to make insertions after the filter is instantiated with an adversarially selected data set), the structure can be made secure by adding a per-representation salt. In the mutable case, a combination of keeping the representation private, swapping hash functions for secretly keyed primitives, and

thresholding (not permitting more insertions to the structure based on the Hamming weight of the bit array) is necessary to secure the structure.

More recently, Filić et al., in [23], analyzed both the Bloom filter and Cuckoo filter [24] (another approximate set-membership PDS) from a provable security perspective. This paper examined both correctness and *privacy* (with respect to hiding the underlying set of elements for a particular representation) for these structures. While Naor and Yogev [19] and Clayton et al. [21] employ a game-based notion of security, this work uses a simulation-based approach. Nonetheless, similar to findings of these previous works, Filić et al. show that swapping the usual hash functions in these structures for keyed primitives (like a pseudorandom function) result in structures that are adversarially robust in terms of both correctness and privacy.

2.3 HyperLogLog

The HyperLogLog (HLL) 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). For a full description of the HLL we point to the original paper [3]. The HLL (including its variants [25], [26]) is widely used to determine the cardinality of a set in settings where adversaries have strong incentives to manipulate the reported answer, such as computing Facebook's distinct visitor count [27], Google's PowerDrill platform [28], network switching [29], and DoS attack detection [30].

Patterson and Raynal [31] 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. They do so in a simulation security based paradigm, which inspired the security framework in Filić et al. [23].

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 [32]. Patterson and Raynal point out this setting in unrealistic, but nonetheless improve the attack in this model. Further, the privacy properties of HLL were examined in [33] with negative results. The use of salts was suggested as a possible mitigation, but no formal analysis of this suggestion was given. It remains an open problem to have a provable security treatment of the privacy properties of HLL with respect to hiding the elements of the underlying set.

2.4 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. Compact frequency estimators are used in the database systems [34], [35], machine learning contexts [36], [37], [38], and networking tools [39], [40], [41]. For a comprehensive survey on the applications of Count-min sketch, one of the most popular compact frequency estimators, we point to [42]. Correctness guarantees of compact frequency estimators assume that the data stream does not depend on the internal randomness of the structure, however this assumption is often not valid in real world deployments.

As previously stated, Clayton et al. were the first to examine compact frequency estimators from a provable security perspective [21]. They specifically examined the Count-min sketch [4] 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.

Continuing this line of work, Markelon et al. [43] analyzed compact frequency estimators in adversarial environments with a provable security style treatment. They analyzed both the Count-min sketch and HeavyKeeper [44], and their results were overwhelmingly negative in all cases. That is, even in the setting in which the internals of the structure were kept private and a secretly keyed primitive was used in place of the usual hash functions, efficient attacks were found that cause large frequency estimation errors on arbitrary elements. We present a comprehensive review of these findings for the Count-min sketch in the case study in the next subsection. Further, there have been a number of works from the streaming data community that have analyzed the adversarial robustness of compact frequency estimators, with similar negative results [45], [46], [47], [48]. Lastly, a number of papers have used compact frequency estimators for privacy preserving schemes [49], [50], [51], but it remains an open area of interest to study the privacy properties of the structures themselves from a provable security perspective.

2.5 Case Study: Count-min sketch

We summarize findings of Markelon et al. [43] regarding the Count-min sketch in this case study.

The Count-min Sketch Structure.

We begin by giving a pseudocode description of the Count-min sketch (CMS) in Figure 1. We (as the authors in [43]) use the syntax for data structures first presented in [21]. The syntax first fixes sets of data objects, responses, and keys. Then it defines a set of allowed queries, each a function from data representation to responses; and a set of allowed updates, each a function from data representation to data representation. A concrete data structure is then a tuple of a representation algorithm REP, a query evaluation algorithm QRY, and an update algorithm UP. The REP

$\underline{\text{REP}_K(\mathcal{S})}$

1 : $M \leftarrow \text{zeros}(k, m)$
2 : **for** $x \in \mathcal{S}$
3 : $\quad M \leftarrow \text{UP}_K(M, \mathsf{up}_x)$
4 : **return** M

$\underline{\text{UP}_K(M, \mathsf{up}_x)}$

1 : $(p_1, \ldots, p_k) \leftarrow R(K, x)$
2 : **for** $i \in [k]$
3 : $\quad M[i][p_i] \mathrel{+}= 1$
4 : **return** M

$\underline{\text{QRY}_K(M, \mathsf{qry}_x)}$

1 : $c \leftarrow \infty$
2 : $(p_1, \ldots, p_k) \leftarrow R(K, x)$
3 : **for** $i \in [k]$
4 : $\quad$ **if** $c > M[i][p_i]$
5 : $\quad\quad c \leftarrow M[i][p_i]$
6 : **return** c

Fig. 1: A possibly keyed count-min sketch structure, $\text{CMS}[R, m, k]$ admitting point queries for any $x \in \mathcal{U}$. The parameters are integers $m, k \geq 0$, and a keyed function $R : \mathcal{K} \times \mathcal{U} \rightarrow [m]^k$ that maps data-object elements (encoded as strings) to a vector of positions in the array M. A concrete scheme is given by a particular choice of parameters.

algorithm instantiates a new concrete instance of a structure with a possible initial data collection. The QRY algorithm answers queries with responses on the structure's representation of its underlying data collection, in accordance with the allowed set of queries and responses. The UP algorithm handles updates to the structure, such as inserting new elements or deleting existing elements. Each algorithm is allowed to take a key to separate secret randomness from any public randomness used to instantiate the structure. The common case of unkeyed structures is captured by setting the keyspace to the empty-set. Given this syntax, we can use it to not only instantiate concrete structures, but also define security notions for those structures. A detailed overview of this syntax is given in Appendix B.

An instance of a CMS structure is initialized as a $k \times m$ matrix M of zero-valued counters, and a mapping R between the universe $\mathcal{U}$ of elements and $[m]^k$. An element x is inserted into the CMS representation by computing $R(K, x) = (p_1, p_2, \ldots, p_k)$, and then adding 1 at each of the counters $M[i][p_i]$ to which x maps. In the original CMS paper [4], it is specified that $(p_1, \ldots, p_k) = (h_1(x), \ldots, h_k(x))$ where the hash functions are sampled at initialization time from a family H of pairwise-independent hash functions. The authors in [43] generalize this to the function referred to as R to not only make the presentation cleaner, but also allow the mapping to depend on secret randomness, namely a key K.

The point query $\text{QRY}(\mathsf{qry}_x)$ returns $\hat{n}_x = \min_{i \in [k]}\{M[i][p_i]\}$, where we take $\hat{n}_x$ to be the returned estimated frequency of x and n_x to be the actual frequency of x. Further, we often shorten this to just $\text{QRY}(x)$ for the sake of brevity. The analysis in [43] considers the insertion-only model, as in the adversarial setting there is no way to enforce the so-called *honest deletion policy*. The policy states that only elements that have been inserted and have a frequency greater than zero may be deleted. There is no way to ensure an adversary adheres to this. In the insertion-only case, it must be that $\hat{n}_x \geq n_x$. This is intuitive, because the CMS estimate on a query for element x selects a counter in some row that x maps to, which contains a value that is of all the insertions of x and all the insertions of other elements that collide with x on that counter. The CMS estimate inherently minimizes this collision noise by choosing the counter with the minimum value.

We can also say, by analysis in [4] that for any $\epsilon, \delta \geq 0$, any $x \in \mathcal{U}$, and any stream $\vec{S}$ (over $\mathcal{U}$) of length N, it is guaranteed that $\Pr[\hat{n}_x - n_x > \epsilon N] \leq \delta$ when: (1) $k = \lceil \ln \frac{1}{\delta} \rceil$, $m = \lceil \frac{e}{\epsilon} \rceil$, and (2) $R(K, x) = (h_1(K \,\|\, x), h_2(K \,\|\, x), \ldots, h_k(K \,\|\, x))$ for h_i that are uniformly sampled from a pairwise-independent hash family H. This guarantee does not depend on anything, except for the length of stream N. However, as [43] points out, this guarantee only holds in the non-adversarial setting. In realty, there is the implicit requirement that the stream and the queried element x are independent of the internal randomness of the structure (i.e., the coins used to sample the h_i). Equivalently said, the stream $\vec{S}$ and the queried element x are determined before the random choices of the structure are made – that is, adversaries are strictly non-adaptive.

The Frequency Error Attack Model.

We present the formal ERR-FE attack model that is given in [43] in Figure 2. The attack model parameters u, v determine whether the adversary $\mathcal{A}$ is given the secret key K and the internal state of the representation repr, respectively. That is, we model settings in which the representation is kept private or is continuously viewable by the adversary. Further, we consider both the keyed hash and unkeyed hash settings. Considering all possible values of u, v gives us a total of four settings that the experiment encodes.

The attack experiment starts by initializing an empty data-object $\vec{S}$, and randomly selecting a key K for the REP, UP, QRY algorithms. Next, an initial representation repr of the empty $\vec{S}$ is computed. The adversary is provided a target $x \in \mathcal{U}$, and given access to oracles that allow it to update the current representation (**Up**) and to make any of the queries permitted by the structure (**Qry**). The adversary (and, implicitly, REP, UP, QRY) is provided oracle access

$\mathbf{Atk}^{\text{err-fe[u,v]}}_{\Pi,\mathcal{U}}(\mathcal{A})$

$\vec{S} \leftarrow \emptyset; K \twoheadleftarrow \mathcal{K}$
$\mathsf{repr} \twoheadleftarrow \text{REP}_K(\vec{S})$
$\mathsf{kv} \leftarrow \top; \mathsf{rv} \leftarrow \top$
if $u = 1$: $\mathsf{kv} \leftarrow K$
if $v = 1$: $\mathsf{rv} \leftarrow \mathsf{repr}$
$x \twoheadleftarrow \mathcal{U}$
$\text{done} \twoheadleftarrow \mathcal{A}^{\mathbf{Hash},\mathbf{Up},\mathbf{Qry}}(x, \mathsf{kv}, \mathsf{rv})$
$n_x \leftarrow \mathsf{qry}_x(\vec{S})$
$\hat{n}_x \leftarrow \text{QRY}_K(\mathsf{repr}, \mathsf{qry}_x)$
return $|\hat{n}_x - n_x|$

$\mathbf{Up}(\mathsf{up})$

$\mathsf{repr}' \twoheadleftarrow \text{UP}_K(\mathsf{repr}, \mathsf{up})$
$\vec{S} \leftarrow \mathsf{up}(\vec{S})$
$\mathsf{repr} \leftarrow \mathsf{repr}'$
if $v = 0$: **return** $\top$
return repr

$\mathbf{Qry}(\mathsf{qry})$

return $\text{QRY}_K(\mathsf{repr}, \mathsf{qry})$

$\mathbf{Hash}(X)$

if $X \notin \mathcal{X}$: **return** $\perp$
if $H[X] = \perp$
  $H[X] \twoheadleftarrow \mathcal{Y}$
return $H[X]$

Fig. 2: The ERR-FE (ERRor in Frequency Estimation) attack model. The attack model returns the absolute difference between the true frequency n_x of an adversarially chosen $x \in \mathcal{U}$, and the estimated frequency $\hat{n}_x$.

to a random oracle $\mathbf{Hash}\colon \mathcal{X} \to \mathcal{Y}$, for some structure-dependent sets $\mathcal{X}, \mathcal{Y}$. The output of the experiment is the absolute difference between the true frequency of x and the structure's reported estimated frequency of x; i.e., $|\hat{n}_x - n_x|$.

The ERR-FE attack model is presented as an attack model, rather than a traditional security experiment. This is because the authors give attacks that created significant frequency estimation error (far higher than the non-adaptive bound) in all settings. Therefore, they do not prove adversarially robustness, and instead they explore lower bounds on the error created by their attacks. The error created by the attacks is reported as a random variable labeled Err in terms of the adversary's resource budget (that is q_Q calls to **Qry**, q_U calls to **Up**, and q_H calls to **Hash** in the ROM.). We next present the attacks from [43].

Insecurity in All Settings.

The authors in [43] give attacks that render large frequency estimation errors in all settings defined in the ERR-FE attack model. We summarize their findings for the CMS here. Note, the authors use the term "public representation setting" and "private representation setting" when $v = 1$ and $v = 0$, respectively, in the ERR-FE attack model. Likewise, they use the term "public hash setting" and "private hash setting" when $u = 1$ and $u = 0$, respectively, in the ERR-FE attack model. Hash functions are made "private" by keying them with a (non-empty) randomly generated secret key. We will use this verbiage as well.

The authors find that a necessary and sufficient condition to create an overestimation error for the CMS on target x is to find a cover set for x. A cover set for x (for a CMS with parameters m, k and key K) is a set of elements $\{y_1, ..., y_k\}$ such that $\forall i \in [k]: R(K, x, i) = R(K, y_i, i)$ and $\forall i \in [k]: y_i \neq x$. In other words, a set of elements distinct from x that collide on every counter that x maps to in the structure.

Once the cover is found, it can be repeatedly inserted, through $\mathbf{Up}(y_i)$ calls on $y_i \in \{y_1, ..., y_k\}$ (an equal amount on each y_i to maximize error). The challenge in all the settings is to find a small cover (with size $\leq k$) using as few resources as possible. Specifically, an attacker wants to minimize the number of insertions it uses to find a cover. This allows the attacks to maximize error, by using as much of its allowed insertions (its **Up** budget q_U) to insert the cover, which causes frequency estimation error on the target x. In turn, it is expected that frequency estimation error $\geq \frac{k}{q'_U}$ is achieved, where q'_U is the number of insertions available to the adversary once a small cover set is found.

We give the public hash setting attack from [43] in Figure 3. The attack never makes use of a view of the representation, so the result is the same if the representation is public or private. The authors provide analysis that shows that the cover set that is found in this attack will be tightly bound to size k. Thus, with sufficient **Hash** query budget, q_H, this attack will achieve $\mathbb{E}[\mathsf{Err}] \geq \lfloor \frac{q_U}{k} \rfloor$. Further, it is shown that the expected number of calls to **Hash** is tightly bound to mH_k, where H_k is the k^{th} harmonic number.

The authors also give an attack in the public representation and private hash setting, where a cover is found by observing elements that cover the counters of the target via the view of the representation and a strategy random insertions. In this case, the authors show that $\mathbb{E}[\mathsf{Err}] \geq \lfloor \frac{q_U - mH_k}{k} \rfloor$.

Most surprisingly, the authors also present an efficient attack in the private representation and private hash setting. The attack follows a three-step algorithm that first finds a cover set using only the **Up** oracle and the **Qry** oracle, then reduces the size of this initial cover to be of size at most k), and finally performs the repeated insertions strategy to maximize error. This attack achieves expected error $\mathbb{E}[\mathsf{Err}] \geq \lfloor \frac{q_U}{k} - mH_k \rfloor$. For full details on these attacks, we again point to the original paper [43].

2.6 Open Problems

Towards a Meaningful Notion of Security for Compact Frequency Estimators.

As shown, the CMS is insecure for the ERR-FE notion of security, even when keeping the internals of the structure private and swapping the usual choice of hash functions for a secretly keyed primitive. This is a surprising result, as use of private representations and private hash functions works

```
CoverAttack^{Hash,Up,Qry}(x, K, repr)
1 : cover ←FindCover^{Hash}(x, K)
2 : until q_U Up-queries made:
3 :   for e ∈ cover: Up(e)
4 : return done

FindCover^{Hash}(x, K)
1 : cover ← ∅; found ← False
2 : I ← ∅; tracker ← zeros(k)
3 : // R(K, x)[i] = Hash(⟨i, K, x⟩)
4 : (p_1, p_2, ..., p_k) ← R(K, x)
5 : while not found
6 :   if q_H Hash-queries made
7 :     return ∅
8 :   y ↞ U \ (I ∪ {x})
9 :   I ← I ∪ {y}
10 :  (q_1, q_2, ..., q_k) ← R(K, y)
11 :  for i ∈ [k]
12 :    if p_i = q_i and tracker[i] ≠ 1
13 :      cover ← cover ∪ {y}
14 :      tracker[i] = 1
15 :  if sum(tracker) = k
16 :    found ← True
17 : return cover
```

Fig. 3: Cover Set Attack for the CMS in public hash function setting. We use $R(K, x)$ to mean $(\mathbf{Hash}(\langle 1, K, x\rangle), \mathbf{Hash}(\langle 2, K, x\rangle, \ldots, \mathbf{Hash}(\langle k, K, x\rangle)))$. The attack is parameterized with the update and **Hash** query budget q_U and q_H, respectively.

in achieving meaningful notions of security for a wide variety of other structures, as shown in the above survey subsections. The information gained from a CMS query response leaks enough information for an attacker to create a large amount of error. Further, the authors in [43] also show that the HeavyKeeper is insecure, and it is not hard to see that the same general technique used here would lead to similar attacks against other compact frequency estimators, like the Count sketch [52] or the XY-Sketch [53].

So we are left to ask – is there some technique that could, be leveraged to provide a meaningful notion of security? To this end, in [21] a provably secure CMS with a private representation and private hash function was provided. However, the authors assumed a version of CMS that disallows updates after a certain structural occupancy threshold is reached. It is unclear if any implementations actually do this, or, more generally, how tenable it would be in practice.

In [43] the authors propose a new structure called the CountKeeper. The attacks given in [43] are less effective and more resource intensive when mounted against CountKeeper, compared to the CMS and HeavyKeeper. Further, the structure gives the ability to flag suspicious (adversarial) estimates, thus potentially providing better practical adversarial robustness. However, the CountKeeper does not provide a provable notion of security – something that we greatly desire.

Barring the creation of a structure whose query responses do not leak information about query response error being created (that can be then used to amplify this error), it seems unlikely any structure can satisfy a notion of security given in the ERR-FE attack model. Thus, a possible avenue for progress could be to investigate a notion of security that reflects the overwhelming use of compact frequency estimators – that of identifying the most frequent elements of a data collection or stream. The general goal is known (in the literature) as the top-K problem [54], the heavy hitters problem [55], or the hot items problem [56]. It is not possible to solve these problems exactly in space less than linear of the data collection [57]. However, compact frequency estimators are commonly used along with a small data structure (such as heap) placed "on top" of them to solve approximate versions of these problems [44], [58], [59].

A security notion for approximate frequent element identification could capture the ability to identify true frequent elements in an adversarially generated stream, without mistakenly reporting non-frequent elements. A possible strategy to go about doing this would be to design a structure that stores elements in a separate stash once their estimated frequency exceeds a threshold that is some fraction of the true frequent element definition. Then, by exactly tracking the insertions of elements in this stash, one can ensure that a particular element is a genuine frequent element, rather than an adversarially produced one. Alternatively, one could experiment with thresholding, like in [21], but on the querying side, instead of the insertion side. One could report all query responses as 0 until they reach a certain threshold (near that of a true frequent element), thus preventing the early discovery of cover sets. Cover sets have been shown in [43] to be a necessary component to create frequency estimation errors in many compact frequency estimators. Therefore, making it resource intensive for an attacker to identify the existence of a cover set can limit their ability to create error. We leave full exploration of these ideas to future work.

More Structures to Explore.

Singh et al. give a comprehensive overview of numerous probabilistic data structures [60], which are perhaps

good targets for a provable security style treatment. We highlight two that are widely used, but lack concrete results around adversarial robustness.

The quotient filter [61] is another approximate set-membership structure. It is a hash table based construction where each entry only consists of a fingerprint of the item being stored and some additional meta-data bits. Quotient filters have been used for privacy-preserving authentication schemes [62], deep packet inspection [63], and duplicate detection over large streams [64]. An examination of the adversarial correctness and privacy is necessary to illuminate if the quotient filter is still appropriate for these applications in the adversarial setting.

Locality-sensitive hashing [65] is a probabilistic data structure that can be used to approximate similarity search measurements. It solves the approximate nearest-neighbor search problem by maximizing hash collisions between similar input elements – in turn, hashing similar elements to the same bucket in the structure. The structure is widely used in computational biology, audio and video fingerprinting, and parallel computing. For a survey on the applications of locality-sensitive hashing, we point the reader to [66]. It would be interesting to explore how the presence of an adaptive adversary could disrupt the similarity matching ability of the structure.

3 Fast Average-Case Runtime Data Structures

3.1 Hash Tables

Hash tables are a data structure that map *keys* to *values* and supports insertions, deletions, and look-ups. They are generally stored in memory as m buckets, where each bucket contains a linked-list of key-value pairs. To map key-value pairs to buckets, a hash function that maps from the key-space to $[m]$ is computed on the key and the key-value pair is stored at the computed bucket index. We formalize this description of hash tables in our case study in 3.3. Assuming the hash function has good collision resistance properties, the amortized average case complexity of insertions, deletions, and look-ups is $\Theta(1)$. For these efficiency reasons, hash tables are widely used in many applications such as implementing associative arrays [67] and sets [68] in many programming languages, in cache systems [69], as well as for database indexing [70].

However, 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. This assumption does not hold in the adversarial case, when insertions may, in fact, depend on the hash function. 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. We will summarize the prior work below, and then provide a case study to explore how one could examine hash tables through a provable security paradigm.

Crosby and Wallach [71] investigate denial-of-service (DoS) attacks via complexity attacks against a number of applications that internally use hash tables to process and store data. Specifically, 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 [72] to fail by overloading the system, causing it to drop all network traffic for a large period of time. The authors suggest that swapping weak hash functions (like the XOR hash function used in Bro) for universal hash functions [73] prevents the type of attacks they describe. However, this is not formally proved.

Similarly, Klink and Walde [74] 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. [75] 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 [76], leading to efficient algorithms for generating arbitrary multi-collisions in all cases. SipHash [77], 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 [78], (to our knowledge) no formalization concerning the provable security of these secretly keyed hash tables exist.

Further, in [79] complexity attacks were presented against two common flow-monitoring systems that use hash tables as part of their core algorithm, greatly increasing the look-up latency of these systems. A hash function the authors believed to be able to prevent these attack was proposed, however little has been done to validate such claims. Moreover, Linux Netfiler [80] attempts to defend against complexity attacks by instantiating each new hash table with a fresh random salt. Yet, in [81] it was shown that remote timing attacks may stultify the use of the salt. If the salt is of small enough length, a brute force attack can be mounted to discover the particular salt used, and then the standard attack can be mounted.

3.2 Skip Lists

A skip list [5] 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 ($\Theta(\log n)$ for all operations). Unlike all the previous structures we have discussed, the random choices of the structure (e.g., the choice of hash functions for PDS or hash tables) are not fixed after instantiation time in skip lists. 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. For a full description of the structure and its algorithms, we again point to the original paper [5]. Skip lists have been used to efficiently process time-series forecasting queries [82], for peer-to-peer asynchronous video streaming[83], and in the construction of verifiable dictionaries [84] (see Section 4).

The original skip list paper [5] 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 they 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. 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 [85] 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.

3.3 Case Study: Hash Tables

$\textsc{Rep}_K(\mathcal{S})$

1 : $T \leftarrow \mathsf{L} \times m$
2 : **for** $(k, v) \in \mathcal{S}$
3 : $\quad T \leftarrow \textsc{Up}_K(T, \mathsf{up}_{(k,v)})$
4 : **return** T

$\textsc{Up}_K(T, \mathsf{up}_{(k,v)})$

1 : $c \leftarrow \textsc{Qry}_K(T, \mathsf{qry}_{(k,v)})$
2 : **if** $c \neq \star$
3 : $\quad \textsc{Up}_K(T, \mathsf{del}_{(k)})$
4 : $i \leftarrow R(K, k)$
5 : $T[i].\mathsf{insert}((k, v))$
6 : **return** T

$\textsc{Up}_K(T, \mathsf{del}_k)$

1 : $c \leftarrow \textsc{Qry}_K(T, \mathsf{qry}_{(k)})$
2 : **if** $c \neq \star$
3 : $\quad i \leftarrow R(K, k)$
4 : $\quad T[i].\mathsf{remove}((k, v))$
5 : **return** T

$\textsc{Qry}_K(T, \mathsf{qry}_k)$

1 : $a \leftarrow \star$
2 : $i \leftarrow R(K, k)$
3 : $b \leftarrow T.\mathsf{find}(k)$
4 : **if** $b \neq \text{null}$
5 : $\quad a \leftarrow b$
6 : **return** a

Fig. 4: A possibly keyed hash-table structure $\text{HT}[R, m]$ admitting insertions, deletions, and queries for any $k \in \mathcal{U}_\kappa$ and its associated value v. The parameters are an integer $m \geq 0$, and a keyed function $R : \mathcal{K} \times \mathcal{U}_\kappa \rightarrow [m]$ that maps the key part of key-value pair data-object elements (encoded as strings) to a position in the one of the table buckets $\boldsymbol{T}$. A concrete scheme is given by a particular choice of parameters. Each bucket contains a simple linked list L equipped with its usual operations. If an item is not contained in the map, the distinguished symbol $\star$ is returned.

The Hash Table Structure. We begin by giving a pseudocode description of a hash table (HT) in the syntax of [21] in Figure 4. An instance of HT consists of m buckets, each containing an (initially empty) linked list L, and a mapping R between the universe $\mathcal{U}_\kappa$ of keys and $[m]$.

A key-value pair (k, v) is inserted to the HT representation by computing $R(K, k){=}i$ and traversing to this i-th bucket. We then check if the pair is already in the linked list L stored there and delete the prior mapping if this is the case. Finally, we insert the current pair into L. Likewise, a key is deleted by searching in the bucket it maps to and removing the key and its associated value from the linked list L in the bucket if this pair exists there. Traditionally, it is assumed that $i = h(x)$, where h is a fast to compute hash function with good (enough) collision resistance properties. However, we generalize here to make the exposition cleaner, and to allow for the mapping to depend upon secret randomness (i.e., a key K). To query for a key for its associated value, the algorithm $\textsc{Qry}(\mathsf{qry}_k)$ searches the bucket k maps to and returns the key-value pair if it exists there; otherwise we return the distinguished null symbol $\star$.

Recall the average time for HT insertions, deletions, and queries is constant. However, this is only when the data inserted into the structure does not depend on the choice of hash function used. This is not something we can assume to be the case in the adversarial setting. We will explicitly demonstrate this with a simple complexity attack, but first we will give a formal notion of adversarial expected performance that can be used to prove a particular HT construction secure or reason about attacks against an insecure construction.

A Notion of Adversarial Performance.

We define a notion of hash table adversarial run time performance in Figure 5, to reason about the expected performance of hash tables when the data they hold may depend arbitrarily on the choice of hash function used to distribute data to buckets. We call this the **HT-Err** experiment. The experiment parameters u, v determine whether the adversary $\mathcal{A}$ is given K and repr, respectively (as in our CMS case study in 2.5). For our purposes, we will only discuss situations when the representation is kept private and consider only the keyed and unkeyed settings. As before, we make hash functions "private" by keying them with a (non-empty) randomly generated secret key.

The experiment starts by initializing an empty data-object $\mathcal{S}$, and randomly selecting a key K for the $\textsc{Rep}, \textsc{Up}, \textsc{Qry}$ algorithms. Next, an initial representation repr of the empty $\mathcal{S}$ is computed. The adversary is given access to oracles that allow it to update the current representation (**Up**) — in effect, to control the data inserted into the table — and to make any of the queries permitted by the structure (**Qry**). Note that when $v = 0$ (which we enforce to be true for the rest of this discussion), the **Up**-oracle leaks nothing about updated representation, so that it remains "private" throughout the experiment. The adversary (and, implicitly, $\textsc{Rep}, \textsc{Up}, \textsc{Qry}$) is provided oracle access to a random oracle **Hash** : $\mathcal{X} \rightarrow \mathcal{Y}$, for sets $\mathcal{X} = \{0, 1\}^*, \mathcal{Y} = [m]$.

We take $\mathsf{count}(\cdot)$ to be a function that returns the number of items that need to be visited in a linked list L of a bucket during a call to one of the operations on the hash table. For illustrative purposes, we designed the experiment such that the adversary's job is as hard as possible. That is, at the end of the experiment, when all insertions are finished,

$\mathbf{Exp}^{\text{ht-err[u,v]}}_{\Pi,\mathcal{U}}(\mathcal{A})$

$\mathcal{S} \leftarrow \emptyset$
$K \twoheadleftarrow \mathcal{K}$
$\mathsf{repr} \twoheadleftarrow \text{REP}_K(\mathcal{S})$
$\mathsf{kv} \leftarrow \top; \mathsf{rv} \leftarrow \top$
if $u = 1 : \mathsf{kv} \leftarrow K$
if $v = 1 : \mathsf{rv} \leftarrow \mathsf{repr}$
$\text{done} \twoheadleftarrow \mathcal{A}^{\mathbf{Hash},\mathbf{Up},\mathbf{Qry}}(\mathsf{kv}, \mathsf{rv})$
$a \leftarrow 0$
for $s \in \mathcal{S}$
  $a \leftarrow a + \mathsf{count}(\text{QRY}_K(\mathsf{repr}, \mathsf{qry}_s))$
return $[a = \frac{|\mathcal{S}|^2 + |\mathcal{S}|}{2} \wedge |S| = q_U]$

$\mathbf{Up}(\mathsf{up})$

$\mathsf{repr}' \twoheadleftarrow \text{UP}_K(\mathsf{repr}, \mathsf{up})$
$\mathcal{S} \leftarrow \mathsf{up}(\mathcal{S})$
$\mathsf{repr} \leftarrow \mathsf{repr}'$
if $v = 0$: **return** $\top$
return repr

$\mathbf{Qry}(\mathsf{qry})$

return $\text{QRY}_K(\mathsf{repr}, \mathsf{qry})$

$\mathbf{Hash}(X)$

if $X \notin \mathcal{X}$: **return** $\perp$
if $H[X] = \perp$
  $H[X] \twoheadleftarrow \mathcal{Y}$
return $H[X]$

Fig. 5: **HT-Err Experiment.** The notion of hash table adversarial run time performance. When experiment parameter $v = 1$ (resp. $v = 0$) then the representation is public (resp. private); when $u = 1$ (resp. $u = 0$) then the structure key K is rendered public (resp. private). The experiment returns 1 if the total number of linked list accesses to query the adversarially generated insertion set $\mathcal{S}$ is the worst-case amount, and 0 otherwise. The **Hash** oracle computes a random mapping $\mathcal{X} \to \mathcal{Y}$ (i.e., a random oracle), and is implicitly provided to REP, UP and QRY.

we output 1 iff querying all the inserted keys touches the maximal number of linked list entries. That is, the adversary must craft a set of keys (and their associated values) such that they all get stored in the same bucket. Formally, we define the advantage of an HT-Err adversary as

$$\mathbf{Adv}^{\text{ht-err}}_{u,v}(A) = \Pr[\mathbf{Exp}^{\text{ht-err[u,v]}}(A) = 1]$$

and take $\mathbf{Adv}^{\text{ht-err}}_{u,v}(t, q_Q, q_U, q_H)$ as the maximum advantage of any HT-Err adversary running in t time steps and making q_Q calls to **Qry**, q_U calls to **Up**, and q_H calls to **Hash** in the ROM.

Insecurity in the Unkeyed Setting.

$\text{FloodAttack}^{\mathbf{Hash},\mathbf{Up},\mathbf{Qry}}(K)$

$n \leftarrow 0$
$\mathcal{S} \leftarrow \emptyset$
until $n = q_U$
  $y \twoheadleftarrow \mathcal{U}_\kappa$
  if $\mathbf{Hash}(K, y) = 1$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{y\}$
    $n \leftarrow n + 1$
for $s \in \mathcal{S}$
  $\mathbf{Up}(\mathsf{up}_{(s,v)})$
return done

Fig. 6: Hash flood attack for the HT in public hash function setting. We flood the first bucket in this particular attack. Further, we assume v takes on some dummy value appropriate for the context in which the attacks occur. The attack is parameterized with the update and **Hash** query budget q_U and q_H.

We give an attack that trivially wins the HT-Err experiment in Figure 6. Given **Up** budget q_U to win the game, we need to make exactly q_U insertions such that they all map to the same bucket. We do this by selecting random $y \in \mathcal{U}_\kappa$ (from the universe of keys in the key-value pairs), then calling $\mathbf{Hash}(K, y)$ and seeing if they map to the target bucket (in this case, the first bucket). Once we have selected exactly q_U keys such that this is the case, we assign them some arbitrary value appropriate for the context of the attack, insert all the pairs, and announce $done$.

The adversary will win the HT-Err experiment game with probability 1 if their **Hash** budget q_H is large enough to compute q_U such keys that all map to a single bucket. We can compute the expected number of call to **Hash** denoted η as

$$E[\eta] = mq_U(1 - \frac{1}{m})$$

from the negative binomial distribution. Having $q_H \gg q_U$ is reasonable in practice, as hash computations are (generally speaking) cheap and can be done entirely offline.

Security in the Keyed Setting.

We now make an informal claim about the security of hash tables in the keyed setting.

Claim 1. *The advantage of an adversary in the keyed setting,* $\mathbf{Adv}^{\text{ht-err}}_{0,0}(t, q_Q, q_U, q_H)$*, is close to that of probability of* q_U *randomly selected elements in the universe mapping to the same bucket (i.e., the non-adaptive bound).*

We leave a formalization of this claim as well as a proof of it for future work. However, we give a few ideas that one can use to sketch a proof and obtain a formal advantage bound. We note that the adversary will never make deletions, as it needs to reserve all of its **Up** budget for strictly insertions to possibly win the game. Further, we can leverage the indistinguishability from random of our keyed primitive (concretely instantiated as, say, a PRF), to replace the insertions of the adversary with those of random elements. In essence, through a series of game hops, we can whittle down the adversary's adaptive ability and roll in the non-adaptive bound. In this case, the non-adaptive bound is simple, as it the probability that q_U items go into 1 of m equally probable buckets or precisely $(\frac{1}{m})^{q_U}$.

3.4 Open Problems

As shown above, provably demonstrating the security of keyed hash tables (with a full theorem statement and rigorous analysis) remains an open problem. It may also be of interest to explore more flexible notions of security that are more permissive in the winning condition of the adversary, as these may align better to real-world deployment needs. Additionally, investigating attacks against hash table variants beyond the simple separate chaining style [86] presented here, such as open addressing [87] and cuckoo hash variants [88], would be valuable. Lastly, it would be vital to assess the feasibility of keying hash tables in real-world deployments, in particular seeing if keyed hash tables were robust to the timing attacks described in [81].

Studying skip lists from a provable security perspective is another open problem. For instance, one could analyze the "secure" splay list construction given in [85]. Moreover, developing a general security model to encompass the entire class of data structures vulnerable to complexity attacks would be beneficial. Doing so would require formally defining this class of structure, and likely identifying other structures in this class.

4 Verifiable Data Structures

4.1 Verifiable Data Structures and Cryptographic Accumulators

The Merkle Tree [89], [90] is the ubiquitous example of a verifiable data structure. It provides efficient and secure verification of potentially large collection of data. It works by ordering the collection of data as leaves in a tree and recursively hashing pairs of data leaves (or, in higher levels of the tree, intermediate hashes) until only a single root hash remains (this being the commitment to the collection). Verification proofs are then given by paths in the tree from the data element being queried to this root hash. They can be verified by a user recomputing the path. The security of the verification property is directly related to the collision resistance property of the hash function used. The representation given by a Merkle tree is linear in space with respect to the underlying data collection, while providing $O(\log n)$ proof size, query time (for inclusion queries), and verification time. Merkle trees can be used to verify the consistency of data between two (or more) mutually distrusting parties, such as done in blockchains [91]. Merkle trees also form the basis of a number of verifiable certificate revocation schemes[7], [8], [9], [10]. A theoretical analysis of verifiable dictionaries using hash trees is given in [92].

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.) [93], [84], [7], [94], [95]. There has also been work done on crafting verifiable data structures which allow for updates. Goodrich and Tammassia [84] give a mutable verifiable dictionary that makes use of hierarchical hashing over skip lists. This structure achieves the same efficiency as immutable tree based constructions. This technique was subsequently used to implement transparency logs for auditing of certificate authorities [96]. Other approaches to mutable verifiable data structures use an abstraction known as dynamic cryptographic accumulators. Cryptographic accumulators were first presented by Benaloh and de Mare [97]. Initially, accumulators allowed a user to verifiably query a fixed set represented by the accumulator without revealing the other elements in the base set. Barić and Pfitzmann [98] generalized the notion of accumulators to provide collision-free constructions that are necessary for many applications. Subsequently, Camenisch and Lyskana [99] defined dynamic accumulators and provided an efficient construction based on the RSA assumption. For a detailed overview of cryptographic accumulators, we point the reader to [100]. Dynamic (i.e., mutable) verifiable dictionaries have been constructed based on cryptographic accumulators [101], [102]. Constructions based on accumulators provide constant proof size and verification time, along with $O(\sqrt{n})$ query and update time.

A formal abstraction for (mutable) verifiable data structures is given in [103]. They also provide definitions for (what they call) "correctness" and "security", that structures of this class must satisfy. Roughly speaking, correctness demands that honestly generated query responses must verify; security requires that dishonestly produced responses will not verify. This formalizes the notion of what is meant by "verifiability". Similarly, Miller et al. [104] present a method for crafting verifiable data structures generically. Specifically, they give a transformation that takes as input a non-verifiable structure (selected from a large class of structures) and outputs a verifiable version of the structure. These works are an advancement in the field. Previously, the study of verifiable data structures was limited to a small subset of all possible data structures of interests, the constructions were largely ad hoc, and the exact security demands were relatively informal.

4.2 Zero-Knowledge Sets

Zero-knowledge sets (ZKS) extend strong privacy definitions to verifiable structures. A ZKS, as first presented by Micali et al. in [105], is an immutable structure that represents a set $\mathcal{S} \subseteq \mathcal{U}$ for some universe $\mathcal{U}$. A query for $x \in \mathcal{U}$ returns the *verifiable* binary response $x \in \mathcal{S}$ or $x \notin \mathcal{S}$ via attaching a proof to the query response, without revealing any other information about $\mathcal{S}$ – including the size of the underlying set $|\mathcal{S}|$.[1] A ZKS construction must satisfy well-defined completeness, soundness, and zero-knowledge properties. Completeness states that any honestly answered query from a well-formed representation can be verified. Soundness states that no adversary can create conflicting answers for any query that both verify. Zero-knowledge captures the property of not discovering anything about the underlying set (including its size) when a response is received to a query (except the answer to the query) through a standard computational zero-knowledge simulation definition.

Chase et al. [106] simplified the constructions of ZKS by introducing a primitive called mercurial commitments. Mercurial commitments extended standard cryptographic commitments by providing soft commitments. Soft commitments allow one to tease a commitment to any value after the time of commitment. That is, they lack the traditional

1. The original work also presents zero-knowledge elementary databases (key, value stores) where a query for some element $x \in \mathcal{U}$ returns a corresponding value $y \in \mathcal{R}$ (for some response space $\mathcal{R}$) such that $x \to y$ (or $\perp$ if key x is not contained in the database). We present the special case of sets, where $\mathcal{R} = \{0, 1\}$.

binding property of cryptographic commitment schemes. These mercurial commitments enable the initial constructions of ZKS to be linear in the size of the set they represent (under the assumption the set is sparse with respect to the universe it is selected from), instead of linear in the size of the universe. The representation grows (i.e., subparts of the structure are generated on the fly) as queries are asked on item that are not in the set to conserve the zero-knowledge property.

Following the original paper, there have been a number of works that present additions and improvements to the original ZKS construction. Gennaro and Micali [107] presented independent-ZKS. Independent-ZKS enforce non-malleability on the structure to prevent a man-in-the-middle style attack, where an adversary could initialize a ZKS correlated with that of some honestly initialized structure. Liskov [108] created the first *mutable* ZKS construction. However, they were unable to retain the soundness properties of the standard ZKS on query responses after updates have been issued, as proofs for items that have been updated may no longer be valid. Catalano et al. [109] introduced q-mercurial commitments, which allowed for compact proofs of non-membership (as compared to existing constructions). Libert and Yung [110] extended this work to allow for the same compactness on proofs of membership as well as non-membership.

4.3 Case Study: Key Transparency

Key Transparency (KT) systems address the challenges of public key distribution in end-to-end encrypted communication platforms. KT systems are crucial for preventing trivial *man-in-the-middle* attacks. Traditionally, verifying the authenticity of another party's public key in secure communication systems required cumbersome processes like physical key exchanges or reliance on third-party authorities. KT systems automate this process (without the need for a trusted third-party), ensuring users that they are receiving the correct public key, or at least one that is consistent with what other users of the service are receiving, while preserving privacy. While KT systems cannot prevent a service provider from misbehaving, they ensure that any misbehavior is quickly detectable. KT systems have not only attracted significant academic interest, as evidenced by studies such as [111], [112], [113], [114], [115], [116], but have also been realized by platforms such as Keybase [117], Zoom [118], Google [119], WhatsApp [120], Apple iMessage [121], and Proton [122]. We will briefly describe how a KT system functions, survey the prior academic research done on KT, and discuss some lessons for researchers learned through examining the history of KT systems.

A KT system is generally operated by the service provider of a particular end-to-end messaging service. The service provider maintains, a privacy-preserving verifiable key directory, that is concretely instantiated as an ordered append-only zero-knowledge set (aoZKS) [113], [116]. This directory maps identifiers in the messaging service (e.g., usernames or phone numbers) to their corresponding public keys. The directory gets updates over short time periods, call *epochs*. During each epoch, the service provider collects fresh key updates and appends them to the key directory. The service provider then posts a signed commitment to the state of the directory at a given epoch to a public bulletin board. This ensures that all users of the system are able to retrieve a consistent per-epoch commitment. Using this commitment, users can then verifiably query the KT system for other users' public keys or verify their own key history. Answers to queries should not reveal information about any other identifier-key pairs held in the directory (except for the one directly being queried). Users can complain out-of-band if an unauthorized change to their key history occurred during the preceding epoch, such as a fraudulent public key being added to the key directory. Further, at each epoch, auditors check to ensure that the commitment and privacy-preserving verifiable map are consistent, and that only key insertions occurred (no deletions are allowed). This auditing process does leak anything about any of the individual identifier-key pairs that exist in the directory, only the number of pairs that exist in the directory, and the number of updates that occurred during any particular epoch. For a full description of KT systems, we point the reader to [116].

CONIKS [111] was the first paper to describe and address the problem of key transparency. The idea was based on prior work on transparency logs for web certificates [123]. They presented a number of informal design goals and a construction that was thought to meet these goals. However, they did not present a formal abstraction of a KT system or a formal specification of the privacy-preserving verifiable key directory primitive. Moreover, their construction was actually insecure for their stated highest-level goal of "non-equivocation". In their system, the service provider did no post a signed commitment to the latest version of the key directory to a public bulletin board, instead opting to include a signed commitment with each query response to each individual user. Therefore, the service provider was able to serve diverging views of the key directory to every user by not providing a way to obtain a consistent commitment to the directory.

SEEMless [113] builds on the work of CONIKS by formalizing a notion of a privacy-preserving verifiable key directory (VKD) and formalizing the primitive it is built upon – the aforementioned aoZKS. SEEMless provides strong and explicit security definitions that a VKD and an aoZKS construction must satisfy, and provably show that their constructions succeed in doing so. They further point out a previously unnoticed privacy leakage vulnerability in CONIKS that their system solves. A number of subsequent papers have taken the core ideas of SEEMless and extended them. Notably, Parakeet [114] and OPTIKS [116] provide more performant versions of the primitives first defined in SEEMless to address scalability issues in real-world KT deployments. Further, ELEKTRA [115] attempt to extend the formalizations provided in SEEMless to easily support users with multiple devices (and thus multiple public keys) in KT deployments.

The history of KT systems in the academic literature highlight a number of crucial cautionary points for applied cryptography researchers. Firstly, it demonstrates the importance of formal security definitions and the provable security paradigm. The original KT system, CONIKS [111], lacked formalization, leading to a number of glaring se-

curity flaws. The provable security paradigm (using well-defined and explicit assumptions) is essential for actually guaranteeing the security of a protocol or primitive. This serves as a reminder of the pitfalls of insufficient security considerations, and the need for rigorous security analysis.

SEEMless [113], attempting to address these flaws, formalized KT systems. In doing so, they had to introduce a new privacy-preserving verifiable data structure, the aoZKS. The aoZKS deviates from the traditional ZKS in a number of ways. Unlike the ZKS (and most other verifiable data structures in the literature), the aoZKS lacks implicit trust assumptions. These traditional structures are presented as three party protocols. The party that instantiates the structure and commitment to the structure (as well as, if allowed, handles updates) is trusted by the client(s), but the responder is not. In KT systems, the party that maintains the structure and responds to queries is the same – the service provider. As a result, KT systems only have the guarantee that afflicted users will be able to detect fraudulent updates, and not outright prevent them. Consequently, aoZKS security definitions focus on the core operations of the data structure (e.g., the update, query, and verify operations), rather than relying on a trust model. This speaks to the need to make security notions general and flexible, allowing application designers to flesh out what the security definitions imply to their specific use case, as opposed to limiting the use of a cryptographic primitive due to unnecessary assumptions.

Next, that aoZKS allows for greater leakage than the traditional ZKS. The traditional ZKS does leak the size of the set it represents. This requires that the representation must be the size of the universe from which the set is selected (or at least capable of expanding to this size, as in [106]). This in impractical for many real-world applications that need to store the representation on real hardware and efficiently respond to queries. For example, the universe of all possible public keys in an end-to-end encrypted chat application is too large to enumerate. Moreover, it is generally not considered sensitive to leak how many people use a specific end-to-end encrypted chat system. Therefore, aoZKS relaxes the privacy definition to allow for leaking the size of the set being representing, while ensuring that the structure leaks nothing about which specific identifier-key pairs exist in the representation and that a query does reveal any identifier-key pair except the one queried. It can be argued that the ZKS is largely a theoretical construction, and thus it is acceptable that it is not feasible to instantiate on real-world infrastructure. However, researchers must consider when their security notions are "too strong" and offer alternatives or ideas on how to craft constructions applicable to real-world deployments.

Finally, we note that KT systems had to address the challenge of preserving soundness (i.e., verifiability) while allowing updates. Recall, Liskov's updatable ZKS [108] lacked soundness guarantees after updates were made to the structure. This lack of soundness is caused by the implicit qualification of *when* a query is made. In other words, a proof verifying a query response may not be valid after an update is made, as that the value corresponding to the proof may have changed. To retain a soundness guarantee, the SEEMless paper encapsulates the aoZKS in a higher abstraction – the VKD. The VKD includes a notion of time (i.e., when updates and queries occur) through epochs. VKDS ensure the soundness of query responses only for the most recent epoch, thereby overcoming this issue of *when* a query is made. This suggests researchers must being willing to add more complexity to their abstractions and constructions (only as strictly necessary), when seeking to conserve security notions in more permissive settings.

5 Conclusion

We examined how the provable security paradigm has been applied (or begs to be applied) to three different classes of data structures. First, we surveyed the literature that examines the adversarial correctness of various compact probabilistic data structures. We also present a case study of the Count-min sketch, summarizing the work of Markelon et al. [43], to provide a detailed overview on how provable security analysis is carried out for structures of this class.

Next, we highlighted the work done in exploring "complexity attacks" against hash tables and skip lists. Several works detail attacks that force worst-case runtime of these structures in real-world applications. Noting a lack of formalization of these attacks and an absence of provable security treatment of proposed countermeasures, we give a case study showing how one might approach analyzing secure hash tables constructions. Along the way, we also highlighted open problems of interest for these two classes of data structure.

Finally, we reviewed the body of research on verifiable data structures. Unlike the previous two classes of data structures, these structures are designed with specific security notions in mind. We concluded with a study on key transparency systems, which use verifiable structures to achieve their desired system goals. We note a mismatch between the needs of key transparency systems, and the security properties that are captured in the traditional verifiable structure literature. This serves as a reminder that the provable security paradigm is not a "one-size-fits-all" solution. Careful consideration is required when crafting security notions and making assumptions about the environments where primitives and protocols will be deployed.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA: MIT Press, third edition ed., 2009.

[2] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[3] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," *Discrete mathematics & theoretical computer science*, no. Proceedings, 2007.

[4] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[5] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Communications of the ACM*, vol. 33, no. 6, pp. 668–676, 1990.

[6] R. Tamassia, "Authenticated data structures," in *Algorithms-ESA 2003: 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003. Proceedings 11*, pp. 2–5, Springer, 2003.

[7] M. Naor and K. Nissim, "Certificate revocation and certificate update," *IEEE Journal on selected areas in communications*, vol. 18, no. 4, pp. 561–570, 2000.

[8] P. C. Kocher, "On certificate revocation and validation," in *Inter-*

national conference on financial cryptography, pp. 172–177, Springer, 1998.

[9] I. Gassko, P. S. Gemmell, and P. 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*, pp. 342–353, Springer, 2000.

[10] A. Buldas, P. Laud, and H. Lipmaa, "Accountable certificate management using undeniable attestations," in *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pp. 9–17, 2000.

[11] M. Shin, C. Straub, R. Tamassia, and D. J. Polivy, "Authenticating web content with prooflets," *Technical report, Center for Geometric Computing, Brown University*, 2002.

[12] D. J. Polivy and R. Tamassia, "Authenticating distributed data using web services and xml signatures," in *Proceedings of the 2002 ACM workshop on XML security*, pp. 80–89, 2002.

[13] I. Mironov, M. Naor, and G. Segev, "Sketching in adversarial environments," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 651–660, 2008.

[14] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1–26, 2008.

[15] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[16] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 1, pp. 131–155, 2011.

[17] B. Goodwin, M. Hopcroft, D. Luu, A. Clemmer, M. Curmei, S. Elnikety, and Y. He, "Bitfunnel: Revisiting signatures for search," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 605–614, 2017.

[18] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang, "On the false-positive rate of bloom filters," *Information Processing Letters*, vol. 108, no. 4, pp. 210–213, 2008.

[19] M. Naor and E. Yogev, "Bloom filters in adversarial environments," in *Annual Cryptology Conference*, pp. 565–584, Springer, 2015.

[20] T. Gerbet, A. Kumar, and C. Lauradoux, "The power of evil choices in bloom filters," in *2015 45th Annual IEEE/IFIP International Conference on dependable systems and networks*, pp. 101–112, IEEE, 2015.

[21] D. Clayton, C. Patton, and T. Shrimpton, "Probabilistic data structures in adversarial environments," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1317–1334, 2019.

[22] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM transactions on networking*, vol. 8, no. 3, pp. 281–293, 2000.

[23] M. Filic, K. G. Paterson, A. Unnikrishnan, and F. Virdia, "Adversarial correctness and privacy for probabilistic data structures," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1037–1050, 2022.

[24] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pp. 75–88, 2014.

[25] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proceedings of the 16th International Conference on Extending Database Technology*, pp. 683–692, 2013.

[26] Y. Chabchoub and G. Heébrail, "Sliding hyperloglog: Estimating cardinality in a data stream over a sliding window," in *2010 IEEE International Conference on Data Mining Workshops*, pp. 1297–1303, IEEE, 2010.

[27] M. Honarkhah and A. Talebzadeh, "Hyperloglog in presto: A significantly faster way to handle cardinality estimation," *Online: https://engineering. fb. com/data-infrastructure/hyperloglog*, 2018.

[28] A. Hall, O. Bachmann, R. Büssow, S. Gănceanu, and M. Nunkesser, "Processing a trillion cells per mouse click," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1436–1446, 2012.

[29] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter, "Evaluating the power of flexible packet processing for network resource allocation," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pp. 67–82, 2017.

[30] Y. Chabchoub, R. Chiky, and B. Dogan, "How can sliding hyperloglog and ewma detect port scan attacks in ip traffic?," *EURASIP Journal on Information Security*, vol. 2014, pp. 1–11, 2014.

[31] K. G. Paterson and M. Raynal, "Hyperloglog: Exponentially bad in adversarial settings," in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pp. 154–170, IEEE, 2022.

[32] P. Reviriego and D. Ting, "Security of hyperloglog (hll) cardinality estimation: Vulnerabilities and protection," *IEEE Communications Letters*, vol. 24, no. 5, pp. 976–980, 2020.

[33] D. Desfontaines, A. Lochbihler, and D. Basin, "Cardinality estimators do not preserve privacy," *Proceedings on Privacy Enhancing Technologies*, vol. 2, pp. 26–46, 2019.

[34] M. Nunberb and A. Shutl, "Redisbloom." https://github.com/RedisBloom/RedisBloom, 2017.

[35] G. Cormode, "Sketch techniques for approximate query processing," *Foundations and Trends in Databases. NOW publishers*, p. 15, 2011.

[36] T. K. R. Medini, Q. Huang, Y. Wang, V. Mohan, and A. Shrivastava, "Extreme classification in log memory using count-min sketch: A case study of amazon search with 50m products," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[37] J. Jiang, F. Fu, T. Yang, and B. Cui, "Sketchml: Accelerating distributed machine learning with data sketches," in *Proceedings of the 2018 International Conference on Management of Data*, pp. 1269–1284, 2018.

[38] P. Talukdar and W. Cohen, "Scaling graph-based semi supervised learning to large number of labels using count-min sketch," in *Artificial Intelligence and Statistics*, pp. 940–947, PMLR, 2014.

[39] Z. Zeng, L. Cui, M. Qian, Z. Zhang, and K. Wei, "A survey on sliding window sketch for network measurement," *Computer Networks*, vol. 226, p. 109696, 2023.

[40] M. Cafaro, I. Epicoco, and M. Pulimeno, "Cmss: Sketching based reliable tracking of large network flows," *Future Generation Computer Systems*, vol. 101, pp. 770–784, 2019.

[41] T. Wellem, Y.-K. Lai, C.-Y. Huang, and W.-Y. Chung, "A flexible sketch-based network traffic monitoring infrastructure," *IEEE Access*, vol. 7, pp. 92476–92498, 2019.

[42] B. Sigurleifsson, A. Anbarasu, and K. Kangur, "The count-min sketch data structure and its uses within computer science," 2019.

[43] S. A. Markelon, M. Filic, and T. Shrimpton, "Compact frequency estimators in adversarial environments," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3254–3268, 2023.

[44] T. Yang, H. Zhang, J. Li, J. Gong, S. Uhlig, S. Chen, and X. Li, "Heavykeeper: an accurate algorithm for finding top-k elephant flows," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 1845–1858, 2019.

[45] M. Hardt and D. P. Woodruff, "How robust are linear sketches to adaptive inputs?," in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pp. 121–130, 2013.

[46] E. Cohen, X. Lyu, J. Nelson, T. Sarlós, M. Shechner, and U. Stemmer, "On the robustness of countsketch to adaptive inputs," in *International Conference on Machine Learning*, pp. 4112–4140, PMLR, 2022.

[47] O. Ben-Eliezer, R. Jayaram, D. P. Woodruff, and E. Yogev, "A framework for adversarially robust streaming algorithms," *ACM Journal of the ACM (JACM)*, vol. 69, no. 2, pp. 1–33, 2022.

[48] E. Cohen, J. Nelson, T. Sarlós, and U. Stemmer, "Tricking the hashing trick: A tight lower bound on the robustness of countsketch to adaptive inputs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 7235–7243, 2023.

[49] F. Zhao, D. Qiao, R. Redberg, D. Agrawal, A. El Abbadi, and Y.-X. Wang, "Differentially private linear sketches: Efficient implementations and applications," *Advances in Neural Information Processing Systems*, vol. 35, pp. 12691–12704, 2022.

[50] D. Mir, S. Muthukrishnan, A. Nikolov, and R. N. Wright, "Pan-private algorithms via statistics on sketches," in *Proceedings of the*

thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 37–48, 2011.

[51] L. Melis, G. Danezis, and E. De Cristofaro, "Efficient private statistics with succinct sketches," in *Proceedings of the NDSS 2016*, 2016.

[52] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theoretical Computer Science*, vol. 312, no. 1, pp. 3–15, 2004.

[53] Y. Liu and X. Xie, "Xy-sketch: on sketching data streams at web scale," in *Proceedings of the Web Conference 2021*, pp. 1169–1180, 2021.

[54] X. Chen, S. Gopi, J. Mao, and J. Schneider, "Competitive analysis of the top-k ranking problem," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1245–1264, SIAM, 2017.

[55] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy hitters in streams and sliding windows," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, IEEE, 2016.

[56] G. Cormode and S. Muthukrishnan, "What's hot and what's not: Tracking most frequent items dynamically," *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, pp. 249–278, 2005.

[57] T. Roughgarden and G. Valiant, "Cs168: The modern algorithmic toolbox lecture #2: Approximate heavy hitters and the count-min sketch," p. 15, 2023.

[58] A. Mandal, H. Jiang, A. Shrivastava, and V. Sarkar, "Topkapi: parallel and fast sketches for finding top-k frequent elements," *NeurIPS*, 2018.

[59] A. Metwally, D. Agrawal, and A. E. Abbadi, "An integrated efficient solution for computing frequent and top-k elements in data streams," *ACM Transactions on Database Systems*, vol. 31, p. 1095–1133, sep 2006.

[60] A. Singh, S. Garg, R. Kaur, S. Batra, N. Kumar, and A. Y. Zomaya, "Probabilistic data structures for big data analytics: A comprehensive review," *Knowledge-Based Systems*, vol. 188, p. 104987, 2020.

[61] A. Pagh, R. Pagh, and S. S. Rao, "An optimal bloom filter replacement.," in *Soda*, vol. 5, pp. 823–829, Citeseer, 2005.

[62] S. Goudarzi, S. A. Soleymani, M. H. Anisi, M. A. Azgomi, Z. Movahedi, N. Kama, H. M. Rusli, and M. K. Khan, "A privacy-preserving authentication scheme based on elliptic curve cryptography and using quotient filter in fog-enabled vanet," *Ad Hoc Networks*, vol. 128, p. 102782, 2022.

[63] M. Al-Hisnawi and M. Ahmadi, "Deep packet inspection using quotient filter," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2217–2220, 2016.

[64] S. Dutta, A. Narang, and S. K. Bera, "Streaming quotient filter: A near optimal approximate duplicate detection approach for data streams," *Proceedings of the VLDB Endowment*, vol. 6, no. 8, pp. 589–600, 2013.

[65] A. Chin, "Locality-preserving hash functions for general purpose parallel computation," *Algorithmica*, vol. 12, no. 2-3, pp. 170–181, 1994.

[66] O. Jafari, P. Maurya, P. Nagarkar, K. M. Islam, and C. Crushev, "A survey on locality sensitive hashing algorithms and their applications," *arXiv preprint arXiv:2102.08942*, 2021.

[67] K. Mehlhorn and P. Sanders, "Hash tables and associative arrays," *Algorithms and Data Structures: The Basic Toolbox*, pp. 81–98, 2008.

[68] J. Blandy, J. Orendorff, and L. F. Tindall, *Programming Rust*. " O'Reilly Media, Inc.", 2021.

[69] Z. István, G. Alonso, M. Blott, and K. Vissers, "A hash table for line-rate data processing," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 2, pp. 1–15, 2015.

[70] J. Zobel, S. Heinz, and H. E. Williams, "In-memory hash tables for accumulating text vocabularies," *Information Processing Letters*, vol. 80, no. 6, pp. 271–277, 2001.

[71] S. A. Crosby and D. S. Wallach, "Denial of service via algorithmic complexity attacks," in *12th USENIX Security Symposium (USENIX Security 03)*, 2003.

[72] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[73] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," in *Proceedings of the ninth annual ACM symposium on Theory of computing*, pp. 106–112, 1977.

[74] A. Klink and J. Walde, "Efficient denial of service attacks on web application platforms," in *28th Chaos Communication Congress*, 2011.

[75] J.-P. Aumasson, D. Bernstein, and M. Boblet, "Hash-flooding dos reloaded: attacks and defenses," in *29th Chaos Communications Congress*, 2012.

[76] A. Appleby, "Smhasher." https://github.com/aappleby/smhasher, 2016.

[77] J.-P. Aumasson and D. J. Bernstein, "Siphash: a fast short-input prf," in *International Conference on Cryptology in India*, pp. 489–508, Springer, 2012.

[78] J.-P. Aumasson, M. Boßlet, and D. J. Bernstein, "Hash-flooding dos reloaded:attacks and defenses." Slides presented at the 2011 Application Security Forum – Western Switzerland, October 2011.

[79] D. Eckhoff, T. Limmer, and F. Dressler, "Hash tables for efficient flow monitoring: Vulnerabilities and countermeasures," in *2009 IEEE 34th Conference on Local Computer Networks*, pp. 1087–1094, IEEE, 2009.

[80] R. Rosen, "Netfilter," *Linux Kernel Networking: Implementation and Theory*, pp. 247–278, 2014.

[81] N. Bar-Yosef and A. Wool, "Remote algorithmic complexity attacks against randomized hash tables," in *International Conference on E-Business and Telecommunications*, pp. 162–174, Springer, 2007.

[82] T. Ge and S. Zdonik, "A skip-list approach for efficiently processing forecasting queries," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 984–995, 2008.

[83] D. Wang and J. Liu, "Peer-to-peer asynchronous video streaming using skip list," in *2006 IEEE International Conference on Multimedia and Expo*, pp. 1397–1400, IEEE, 2006.

[84] M. T. Goodrich and R. Tamassia, "Efficient authenticated dictionaries with skip lists and commutative hashing," tech. rep., Technical Report, Johns Hopkins Information Security Institute, 2000.

[85] E. Nussbaum and M. Segal, "Skiplist timing attack vulnerability," in *International Workshop on Data Privacy Management*, pp. 49–58, Springer, 2019.

[86] W. D. Maurer and T. G. Lewis, "Hash table methods," *ACM Computing Surveys (CSUR)*, vol. 7, no. 1, pp. 5–19, 1975.

[87] J. I. Munro and P. Celis, "Techniques for collision resolution in hash tables with open addressing," in *Proceedings of 1986 ACM Fall joint computer conference*, pp. 601–610, 1986.

[88] X. Li, D. G. Andersen, M. Kaminsky, and M. J. Freedman, "Algorithmic improvements for fast concurrent cuckoo hashing," in *Proceedings of the Ninth European Conference on Computer Systems*, pp. 1–14, 2014.

[89] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the theory and application of cryptographic techniques*, pp. 369–378, Springer, 1987.

[90] R. C. Merkle, "Method of providing digital signatures," Jan. 5 1982. US Patent 4,309,569.

[91] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[92] R. T. N. Triandopoulos, "On the cost of authenticated data structures," in *In Proc. European Symp. on Algorithms*, vol. 2832, Citeseer, 2003.

[93] A. Anagnostopoulos, M. T. Goodrich, and R. Tamassia, "Persistent authenticated dictionaries and their applications," in *International Conference on Information Security*, pp. 379–393, Springer, 2001.

[94] M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen, "Authenticated data structures for graph and geometric searching," in *Cryptographers' Track at the RSA Conference*, pp. 295–313, Springer, 2003.

[95] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine, "A general model for authenticated data structures," *Algorithmica*, vol. 39, p. 21–41, jan 2004.

[96] A. Tomescu, V. Bhupatiraju, D. Papadopoulos, C. Papamanthou, N. Triandopoulos, and S. Devadas, "Transparency logs via append-only authenticated dictionaries," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1299–1316, 2019.

[97] J. Benaloh and M. De Mare, "One-way accumulators: A decentralized alternative to digital signatures," in *Workshop on the*

Theory and Application of of Cryptographic Techniques, pp. 274–285, Springer, 1993.

[98] N. Barić and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees," in *International conference on the theory and applications of cryptographic techniques*, pp. 480–494, Springer, 1997.

[99] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*, pp. 61–76, Springer, 2002.

[100] I. Ozcelik, S. Medury, J. Broaddus, and A. Skjellum, "An overview of cryptographic accumulators," *arXiv preprint arXiv:2103.04330*, 2021.

[101] M. T. Goodrich, R. Tamassia, and J. Hasić, "An efficient dynamic and distributed cryptographic accumulator," in *International Conference on Information Security*, pp. 372–388, Springer, 2002.

[102] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Authenticated hash tables based on cryptographic accumulators," *Algorithmica*, vol. 74, pp. 664–712, 2016.

[103] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Optimal verification of operations on dynamic sets," in *Advances in Cryptology – CRYPTO 2011* (P. Rogaway, ed.), (Berlin, Heidelberg), pp. 91–110, Springer Berlin Heidelberg, 2011.

[104] A. Miller, M. Hicks, J. Katz, and E. Shi, "Authenticated data structures, generically," *ACM SIGPLAN Notices*, vol. 49, no. 1, pp. 411–423, 2014.

[105] S. Micali, M. Rabin, and J. Kilian, "Zero-knowledge sets," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pp. 80–91, IEEE, 2003.

[106] M. Chase, A. Healy, A. Lysyanskaya, T. Malkin, and L. 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*, pp. 422–439, Springer, 2005.

[107] R. Gennaro and S. Micali, "Independent zero-knowledge sets," in *International Colloquium on Automata, Languages, and Programming*, pp. 34–45, Springer, 2006.

[108] M. 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*, pp. 174–198, Springer, 2005.

[109] D. Catalano, D. Fiore, and M. 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*, pp. 433–450, Springer, 2008.

[110] B. Libert and M. Yung, "Concise mercurial vector commitments and independent zero-knowledge sets with short proofs," in *Theory of Cryptography Conference*, pp. 499–517, Springer, 2010.

[111] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "{CONIKS}: Bringing key transparency to end users," in *24th USENIX Security Symposium (USENIX Security 15)*, pp. 383–398, 2015.

[112] J. Bonneau, "Ethiks: Using ethereum to audit a coniks key transparency log," in *International Conference on Financial Cryptography and Data Security*, pp. 95–105, Springer, 2016.

[113] M. Chase, A. Deshpande, E. Ghosh, and H. Malvai, "Seemless: Secure end-to-end encrypted messaging with less trust," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pp. 1639–1656, 2019.

[114] H. Malvai, L. Kokoris-Kogias, A. Sonnino, E. Ghosh, E. Oztürk, K. Lewi, and S. Lawlor, "Parakeet: Practical key transparency for end-to-end encrypted messaging," *Cryptology ePrint Archive*, 2023.

[115] J. Len, M. Chase, E. Ghosh, D. Jost, B. Kesavan, and A. Marcedone, "Elektra: Efficient lightweight multi-device key transparency," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2915–2929, 2023.

[116] J. Len, M. Chase, E. Ghosh, K. Laine, and R. C. Moreno, "Optiks: An optimized key transparency system," *Cryptology ePrint Archive*, 2023.

[117] A. 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.

[118] J. Blum, S. Booth, B. Chen, O. Gal, M. Krohn, J. Len, K. Lyons, A. Marcedone, M. Maxim, M. E. Mou, *et al.*, "Zoom cryptography whitepaper," 2022.

[119] R. Hurst and G. Belvin, "google/keytransparency." https://github.com/google/keytransparency/, 2020.

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

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

[122] T. Göbel and D. Huigens, "Proton key transparency whitepaper," 2024.

[123] B. Laurie, "Certificate transparency," *ACM Transactions on Computing*, vol. 57, no. 10, 2014.

[124] R. Zhang, R. Xue, and L. Liu, "Security and privacy on blockchain," *ACM Computing Surveys (CSUR)*, vol. 52, no. 3, pp. 1–34, 2019.

[125] K. Needels and M. Kwon, "Secure routing in peer-to-peer distributed hash tables," in *Proceedings of the 2009 ACM symposium on Applied Computing*, pp. 54–58, 2009.

APPENDIX A
NOTATIONAL CONVENTIONS

Bitstring and Set Operations.

Let $\{0,1\}^*$ denote the set of bitstrings and let ε denote the empty string. Let $X \,\|\, Y$ denote the concatenation of bitstrings X and Y. When $\mathcal{S}$ is an abstract data-object (e.g., a (multi)set, a list) and e is an object that can be appended (in some understood fashion) to $\mathcal{S}$, we overload the $\|$ operator and write $\mathcal{S} \,\|\, e$.

Let $x \twoheadleftarrow \mathcal{X}$ denote sampling x from a set $\mathcal{X}$ according to the distribution associated with $\mathcal{X}$; if $\mathcal{X}$ is finite and the distribution is unspecified, then it is uniform. Let $[i..j]$ denote the set of integers $\{i, \ldots, j\}$; if $i > j$, then define $[i..j] = \emptyset$. For all $m \geq 2$, let $[m] = \{1, 2, \ldots, m\}$.

Let $\mathcal{A}$ and $\mathcal{B}$ be sets. We take $\mathcal{A} \cup \mathcal{B}$ to be the union of the sets, $\mathcal{A} \cap \mathcal{B}$ to be the intersection of the sets, and $\mathcal{A} \setminus \mathcal{B}$ to be set-theoretic difference of $\mathcal{A}$ and $\mathcal{B}$.

Functions.

Let $\mathrm{Func}(\mathcal{X}, \mathcal{Y})$ denote the set of functions $f : \mathcal{X} \to \mathcal{Y}$. For every function $f : \mathcal{X} \to \mathcal{Y}$, define $\textsc{id}^f : \{\varepsilon\} \times \mathcal{X} \to \mathcal{Y}$ so that $\textsc{id}^f(\varepsilon, x) = f(x)$ for all x in the domain of f. This allows us to use unkeyed hash functions H in situations where, syntactically, a function is required to take a key along with its input.

Arrays and Tuples.

We use the distinguished symbol $\star$ to mean that a variable is uninitialized. By $[\text{item}] \times \ell$ for, $\ell \in N$ we mean a vector of ℓ replicas of item. We use $\mathrm{zeros}(m)$ to denote a function that returns a m-length array of 0s and, likewise, $\mathrm{zeros}(k, m)$ to denote a function that returns an $k \times m$ array of 0s. We index into arrays (and tuples) using $[\cdot]$ notation; in particular, if R is a function returning a k-tuple, we write $R(x)[i]$ to mean the i-th element/coordinate of $R(x)$. If $X = (x_1, x_2, \ldots, x_t)$ is a tuple and $\mathcal{S}$ is a set, we overload standard set operators (e.g., $X \subseteq \mathcal{S}$) treating the tuple as a set; if we write $X \setminus \mathcal{S}$, we mean to remove all instances of the elements of $\mathcal{S}$ from the tuple X, returning a tuple X' that is "collapsed" by removing any now-empty positions.

APPENDIX B
A SYNTAX FOR DATA STRUCTURES

We present (a slightly modified) syntax for data structures first provided by [21]. While originally used to describe a variety of probabilistic data structures, the syntax is appropriately general. A syntactic formalization of data structures in this way not only allows us to elegantly describe numerous data structures, but also craft security definitions that are directly related to the operations the data structure allows. We will do exactly this in our case studies throughout the rest of this work.

We start by fixing three non-empty sets $\mathcal{D}, \mathcal{R}, \mathcal{K}$ of *data objects*, *responses* and *keys*, respectively. Let $\mathcal{Q} \subseteq \mathrm{Func}(\mathcal{D}, \mathcal{R})$ be a set of allowed *queries*, and let $\mathcal{U} \subseteq \mathrm{Func}(\mathcal{D}, \mathcal{D})$ be a set of allowed data-object *updates*. A *data structure* is a tuple $\Pi = (\textsc{Rep}, \textsc{Qry}, \textsc{Up})$, where:

- $\textsc{Rep}\colon \mathcal{K} \times \mathcal{D} \to \{0,1\}^* \cup \{\perp\}$ is a (possibly) randomized *representation algorithm*, taking as input a key $K \in \mathcal{K}$ and data object $\mathcal{S} \in \mathcal{D}$, and outputting the representation $\mathsf{repr} \in \{0,1\}^*$ of D, or $\perp$ in the case of a failure. We write this as $\mathsf{repr} \leftarrow \textsc{Rep}_K(\mathcal{S})$.
- $\textsc{Qry}\colon \mathcal{K} \times \{0,1\}^* \times \mathcal{Q} \to \mathcal{R} \cup \{\perp\}$ is a deterministic *query-evaluation algorithm*, taking as input $K \in \mathcal{K}$, $\mathsf{repr} \in \{0,1\}^*$, and $\mathsf{qry} \in \mathcal{Q}$, and outputting an answer $a \in \mathcal{R}$, or $\perp$ in the case of a failure. We write this as $a \leftarrow \textsc{Qry}_K(\mathsf{repr}, \mathsf{qry})$.
- $\textsc{Up}\colon \mathcal{K} \times \{0,1\}^* \times \mathcal{U} \to \{0,1\}^* \cup \{\perp\}$ is a (possibly) randomized *update algorithm*, taking as input $K \in \mathcal{K}$, $\mathsf{repr} \in \{0,1\}^*$, and $\mathsf{up} \in \mathcal{U}$, and outputting an updated representation repr', or $\perp$ in the case of a failure. We write this as $\mathsf{repr}' \leftarrow \textsc{Up}_K(\mathsf{repr}, \mathsf{up})$.

Allowing each of the algorithms to take a key K permits one to separate (for some security notion) any secret randomness used across data structure operations, from per-operation randomness (e.g., generation of a salt). Note that this syntax admits the common case of *unkeyed* data structures, by setting $\mathcal{K} = \{\varepsilon\}$. Moreover, we can set $\mathcal{K} = \mathsf{priv}$ to be a private key and allow the corresponding public key pub to be a public parameter in the case the data structure relies on asymmetric cryptographic primitives.

Both REP and the UP algorithm can be viewed (informally) as mapping data objects to representations — explicitly so in the case of REP, and implicitly in the case of UP — so we allow UP to make per-call random choices, too.

Note that UP takes a function operating on data objects as an argument, even though UP itself operates on *representations* of data objects. This is intentional, to match the way these data structures generally operate. In a data structure representing a set or multiset, we often think of performing operations such as 'insert x' or 'delete y'. When the set or multiset is not being stored, but instead modeled via a representation, the representation must transform these operations into operations on the actual data structure it is using for storage. This is common for operation on probabilistic data structures.

We also note that the query algorithm QRY is deterministic, which reflects the overwhelming majority of data structures in practice. Allowing QRY to be randomized would allow for a greater degree of syntactic expressiveness, particularly for some data structures that provide privacy guarantees. However, it can make it more difficult to craft correctness properties in that it may be difficult to discern the errors caused by an adaptive adversary versus "intended" error arising from the randomized query algorithm. Care must be taken when both designing structures and defining security properties to ensure issues do not arise from this.

Further, while the syntax from [21] that we present above is highly expressive, it may not capture all data structures one may want to examine. For instance, an *verifiable data structure* [6] may require some changed to the functional definitions, as well as the addition of a *verification function* VFY that would act to verify proofs to query responses. Lastly, we note that the syntax does not support describing (nor is it clear how to easily modify the syntax to be able to support) what could be viewed as distributed data structures, such as blockchains and distributed hash tables. This is not the focus of this work, and we point to the reader [124] and [125], respectively, if interested in such areas.