Compact Frequency Estimators in Adversarial Environments

Sam A. Markelon
University of Florida
Gainesville, FL, USA
smarkelon@ufl.edu

Mia Filić
ETH Zürich
Zürich, Switzerland
mia.filic@inf.ethz.ch

Thomas Shrimpton
University of Florida
Gainesville, FL, USA
teshrim@ufl.edu

ABSTRACT

Count-Min Sketch (CMS) and HeavyKeeper (HK) are two realizations of a compact frequency estimator (CFE). These are a class of probabilistic data structures that maintain a compact summary of (typically) high-volume streaming data, and provides approximately correct estimates of the number of times any particular element has appeared. CFEs are often the base structure in systems looking for the highest-frequency elements (i.e., top-K elements, heavy hitters, elephant flows). Traditionally, probabilistic guarantees on the accuracy of frequency estimates are proved under the implicit assumption that stream elements do not depend upon the internal randomness of the structure. Said another way, they are proved in the presence of data streams that are created by non-adaptive adversaries. Yet in many practical use-cases, this assumption is not well-matched with reality; especially, in applications where malicious actors are incentivized to manipulate the data stream. We show that the CMS and HK structures can be forced to make significant estimation errors, by concrete attacks that exploit adaptivity. We analyze these attacks analytically and experimentally, with tight agreement between the two. Sadly, these negative results seem unavoidable for (at least) sketch-based CFEs with parameters that are reasonable in practice. On the positive side, we give a new CFE (Count-Keeper) that can be seen as a composition of the CMS and HK structures. Count-Keeper estimates are typically more accurate (by at least a factor of two) than CMS for "honest" streams; our attacks against CMS and HK are less effective (and more resource intensive) when used against Count-Keeper; and Count-Keeper has a native ability to flag estimates that are suspicious, which neither CMS or HK (or any other CFE, to our knowledge) admits.

CCS CONCEPTS

• **Security and privacy → Hash functions and message authentication codes**.

KEYWORDS

probabilistic data structures; Count-min sketch; HeavyKeeper; Count-Keeper

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11…$15.00
https://doi.org/10.1145/3576915.3623216

ACM Reference Format:
Sam A. Markelon, Mia Filić, and Thomas Shrimpton. 2023. Compact Frequency Estimators in Adversarial Environments. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623216

1 INTRODUCTION

The use of probabilistic data structures (PDS) has grown rapidly in recent years in correlation with the rise of distributed applications producing and processing huge amounts of data. Probabilistic data structures provide compact representations of (potentially massive) data, and support a small set of queries. The trade-off for compactness is that query responses are only guaranteed to be "close" to the true answer (i.e., if the query were evaluated on the full data) with a certain probability. For example, the ubiquitous Bloom filter [6] admits data-membership queries (*Does element x appear in the data?*). The probabilistic guarantee on the correctness of responses assumes that the data represented by the Bloom filter is independent of the randomness used to sample the hash functions that are used to populate the filter, and to compute query responses. This is equivalent to providing correctness guarantees in the presence of adversarial data sets and queries that are *non-adaptive*, i.e., made in advance of the sampling of the hash functions.

A number of recent works — notably those of Naor and Yogev [25], Clayton, Patton and Shrimpton [8], and Filić et al. [13] — have provided detailed analyses of Bloom filters under *adaptive* attacks; the results are overwhelmingly negative. Paterson and Raynal [26] provided similar results for the HyperLogLog PDS, which can be used to count the number of distinct elements in a data collection [14].

In this work, we focus on PDS that can be used to estimate the number of times any particular element x appears in a collection of data, i.e., the *frequency* of x. Such compact frequency estimators (CFEs) are commonly used in streaming settings, to identify elements with the largest frequencies — so-called *heavy hitters* or *elephants*. Finding extreme elements is important for network planning [12], network monitoring [18], recommendation systems [22], and approximate database queries [2], to name a few applications.

The Count-min Sketch (CMS) [10] and HeavyKeeper (HK) [30] structures are two CFEs that we consider, in detail. The CMS structure has been widely applied to a number of problems outlined above. Details on these applications are thoroughly examined in the survey paper by Sigurleifsson et al. [29]. The HK structure is the CFE of choice in the RedisBloom module [2], a component of the Redis database system [1].

As is the case for Bloom filters, HyperLogLog and other PDS, the accuracy guarantees for CFEs effectively assume that the data they represent were produced by a non-adaptive strategy. Our work

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

Our findings are negative in all cases. No matter the combination of public and private, a well resourced adversary can force CMS and HK estimates to be arbitrarily far from the true frequency. As one example of what this means for larger systems, things that have never appeared in the stream can be made to look like heavy hitters (in the case of CMS), and legitimate heavy hitters can be made to disappear entirely (in the case of HK). This is somewhat surprising in the "private-private" setting, where the attack can only gain information about the structure and its operations via frequency estimate queries. Of course, there are differences in practice: when attacks are forced to be online, they are easier to detect and throttle, so the query-resource terms in our analytical results are likely capped at smaller values than when some or all of an attack can progress offline.

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

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

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

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

The Count-Keeper CFE. We propose a new structure that, roughly speaking, combines equally sized (still compact) CMS and HK structures, and provide analytical and empirical evidence that it reduces the error (by at least a factor of two) that can be induced once a cover set is found. It also requires a type of cover set that is roughly twice as expensive (in terms of oracle queries) to find. Moreover, it can effectively detect when the reported frequency of an element is likely to have large error. In this way we can dampen the effect of the attacks, by catching and raising a *flag* when a cover set has been found and is inserted many times to induce a large frequency error estimation on a particular element.

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

As a side note, we uncovered numerous analytical errors in [30] that invalidate some of their claims about the behaviors of the HK structure. We have communicated with the authors of [30] and contacted Redis, whose RedisBloom library implements HK (and CMS) with fixed, public hash functions (i.e., the internal randomness is fixed for all time and visible to attackers).

In [16], the authors consider adding robustness to streaming algorithms using differential privacy. Meanwhile, Hardt and Woodruff [15], Cohen et al. [9] and Ben-Eliezer et al.[4] have shown that linear sketches (including CMS but not HK) are not "robust" to well-resourced adaptive attacks, when it comes to various L_p-norm estimation tasks, e.g., solving the k-heavy-hitters problem relative

to the L_2-norm. These works are mostly of theoretical importance, whereas we aim to give concrete attacks and results that are (more) approachable for practitioners.

2 DATA STRUCTURE SYNTAX

In this section, we formalize data structures as abstract, syntactic objects, and make explicit the attack model that we consider throughout the paper. Before that, some notational conventions.

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 $\{0,1\}^*$ denote the set of bitstrings and let ε denote the empty string. Let $\mathrm{Func}(\mathcal{X},\mathcal{Y})$ denote the set of functions $f : \mathcal{X} \to \mathcal{Y}$. We use $\star$ 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 $\mathrm{zeros}(k,m)$ to denote a function that returns a $k \times m$ 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.

2.1 Data structures

We adopt the syntax of Clayton et al.[8]. For space reasons, our description is terse; please see [8] for a full discussion of their syntactic choices. Fix non-empty sets $\mathcal{D},\mathcal{R},\mathcal{K}$ of *data objects*, *responses* and *keys*, respectively. Let $\mathcal{F}_q \subseteq \mathrm{Func}(\mathcal{D},\mathcal{R})$ be a set of allowed *queries*, and let $\mathcal{F}_u \subseteq \mathrm{Func}(\mathcal{D},\mathcal{D})$ 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 randomized *representation algorithm*, $\textsc{Qry}\colon \mathcal{K} \times \{0,1\}^* \times \mathcal{F}_q \to \mathcal{R}$ is a deterministic *query-evaluation algorithm*, and $\textsc{Up}\colon \mathcal{K} \times \{0,1\}^* \times \mathcal{F}_u \to \{0,1\}^* \cup \{\perp\}$ is a randomized *representation-update algorithm*.

Allowing each algorithm to take a key $K \in \mathcal{K}$ provides an explicit mechanism for separating randomness used across algorithms and their executions, from per-operation randomness that is local to each algorithm. HeavyKeeper is one example of a data structure with per-operation randomness. In our security model, the key may be secret (i.e., not given to the adversary) or public. Secret keys are an explicit mechanism for reasoning about adversarial correctness when representations are meant to be private. Note that the traditional *unkeyed* data structures are captured by setting $\mathcal{K} = \{\varepsilon\}$. To make clear the semantic differences among their inputs, we write $\mathrm{repr} \twoheadleftarrow \textsc{Rep}_K(\mathcal{S}), a \leftarrow \textsc{Qry}_K(\mathrm{qry})$, and $\mathrm{repr} \twoheadleftarrow \textsc{Up}_K(\mathrm{up})$ for the execution of these algorithms. The abstract insertion function up_x is handled by Up as a concrete action of (say) carrying out certain hashing operations, and incrementing counters indicated by those hashing operations. Side-effects of Up, or cases where the algorithm's behavior does not perfectly match the intended update up, are a potential source of errors that an adversary can exploit. We assume that all data structures admit *frequency estimation queries* (also known as *point queries*) qry_x for each $x \in \mathcal{U}$, defined so that $\mathrm{qry}_x(\mathcal{S})$ returns the number of occurrences n_x of x in data-object $\mathcal{S}$.

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

1 : $\vec{S} \leftarrow \emptyset; K \twoheadleftarrow \mathcal{K}$
2 : $\mathrm{repr} \twoheadleftarrow \textsc{Rep}_K(\vec{S})$
3 : $\mathrm{kv} \leftarrow \top; \mathrm{rv} \leftarrow \top$
4 : **if** $u = 1$: $\mathrm{kv} \leftarrow K$
5 : **if** $v = 1$: $\mathrm{rv} \leftarrow \mathrm{repr}$
6 : $x \twoheadleftarrow \mathcal{U}$
7 : $\mathrm{done} \twoheadleftarrow \mathcal{A}^{\mathbf{Hash},\mathbf{Up},\mathbf{Qry}}(x, \mathrm{kv}, \mathrm{rv})$
8 : $n_x \leftarrow \mathrm{qry}_x(\vec{S})$
9 : $\hat{n}_x \leftarrow \textsc{Qry}_K(\mathrm{repr}, \mathrm{qry}_x)$
10 : **return** $|\hat{n}_x - n_x|$

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

1 : $\mathrm{repr}' \twoheadleftarrow \textsc{Up}_K(\mathrm{repr}, \mathrm{up})$
2 : $\vec{S} \leftarrow \mathrm{up}(\vec{S})$
3 : $\mathrm{repr} \leftarrow \mathrm{repr}'$
4 : **if** $v = 0$: **return** $\top$
5 : **return** repr

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

1 : **return** $\textsc{Qry}_K(\mathrm{repr}, \mathrm{qry})$

$\mathbf{Hash}(X)$

1 : **if** $X \notin \mathcal{X}$: **return** $\perp$
2 : **if** $H[X] = \perp$
3 : $\quad H[X] \twoheadleftarrow \mathcal{Y}$
4 : **return** $H[X]$

Figure 1: the ERR-FE (ERRor in Frequency Estimation) attack model. 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 the absolute difference between the true frequency n_x of an adversarially chosen $x \in \mathcal{U}$, and the estimated frequency $\hat{n}_x$. 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.

2.2 Streaming Data

A *stream* data-object $\vec{S} = e_1, e_2, \ldots$ is a finite sequence of elements $e_i \in \mathcal{U}$ for some universe $\mathcal{U}$. The elements of a stream are not necessarily distinct, and the (stream) frequency of some $x \in \mathcal{U}$ is $|\{i : e_i = x\}|$. From the perspective of the PDS, the stream is presented one element at a time, with no buffering or "look ahead". That is, processing of a stream is performed in order, and the processing of e_i is completed before the processing of e_{i+1} may begin; once e_i has been processed, it cannot be revisited.

2.3 Formal Attack Model

To enable precise reasoning about the correctness of frequency estimators when data streams may depend, in arbitrary ways, on the internal randomness of the data structure, we give a pseudocode description of our attack model in Figure 1. The experiment parameters u, v determine whether the adversary $\mathcal{A}$ is given K and repr, respectively. Thus, there are actually four attack models encoded into the experiment. The adversary is provided a target $x \in \mathcal{U}$, and given access to oracles that allow it to update the current representation (**Up**) — in effect, to control the data stream — and to make any of the queries permitted by the structure (**Qry**). We abuse notation for brevity and write $\mathbf{Up}(e)$ to mean an insertion of e into the structure and $\mathbf{Qry}(e)$ to get a point query on e for some element $e \in \mathcal{U}$. Note that when $v = 1$, the **Up**-oracle leaks nothing about updated representation, so that it remains "private" throughout the experiment. The adversary (and, implicitly, Rep, Up, Qry) is provided oracle access to a random oracle $\mathbf{Hash} : \mathcal{X} \to \mathcal{Y}$, for some structure-dependent sets $\mathcal{X}, \mathcal{Y}$. The output of the experiment is the absolute error between the true frequency n_x of x in the adversarial data stream, and the structure's estimate $\hat{n}_x$ of n_x.

3 COUNT-MIN SKETCH

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

$\textsc{Up}_K(M, \text{up}_x)$
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

$\textsc{Qry}_K(M, \text{qry}_x)$
1 : $(p_1, \ldots, p_k) \leftarrow R(K, x)$
2 : **return** $\min_{i \in [k]} \{M[i][p_i]\}$

Figure 2: 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.**

Figure 2 gives a pseudocode description of the count-min sketch (CMS), in our syntax. An instance of CMS consists of a $k \times m$ matrix M of (initially zero) counters, and a mapping R between the universe $\mathcal{U}$ of elements and $[m]^k$. An element x is added to the CMS representation by computing $R(K, x) = (p_1, p_2, \ldots, p_k)$, and then adding 1 to each of the counters at $M[i][p_i]$. Traditionally, it is assumed that $(p_1, p_2, \ldots, p_k) = (h_1(x), h_2(x), \ldots, h_k(x))$ where the h_i are sampled at initialization from some family H of hash functions, but we generalize here to make the exposition cleaner, and to allow for the mapping to depend upon secret randomness (i.e., a key K).

The point query $\textsc{Qry}(\text{qry}_x)$ returns $\hat{n}_x = \min_{i \in [k]}\{M[i][p_i]\}$. We note that it must be that $\hat{n}_x \geq n_x$. To see this, let $V_x^i = \{y \in \vec{S} \mid y \neq x \text{ and } R(y)[i] = p_i\}$ be the set of elements that "collide" with x's counter in the i-th row. Then we can write $M[i][p_i] = n_x + \sum_{y \in V_x^i} n_y$, where $n_y \geq 0$. Viewed this way, we see that a CMS estimate $\hat{n}_x$ minimizes the "collision noise", i.e., $\hat{n}_x = n_x + \min_{i \in [k]}\{\sum_{y \in V_x^i} n_y\}$.

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 [10]. Implicitly, there is a third requirement, namely (3) the stream and the target x are independent of the internal randomness of the structure (i.e., the coins used to sample the h_i).

4 HEAVYKEEPER

Like CMS, an instance of the HeavyKeeper data structure is parameterized by positive integers k, m, and a function $R\colon \mathcal{K} \times \mathcal{U} \rightarrow [m]^k$; in addition, it is parameterized by real-valued $d \in (0, 1]$, and *fingerprinting function* $T\colon \mathcal{K} \times \mathcal{U} \rightarrow \{0,1\}^n$ for some fixed $n > 0$. The HK structure (see the pseudocode in Figure 3) maintains a $k \times m$ matrix A. However, each $A[i][j]$ holds a pair (fp, cnt), initialized as $(\star, 0)$ where $\star$ is a distinguished symbol. Informally, for a given stream $\vec{S}$, any $z \in \vec{S}$ such that $A[i][j].\text{fp} = T(K, z)$ is an *owner* of this position; there may be more than one such owner at a time, if $T(K, \cdot)$ admits many collisions. Ownership can change as

$\textsc{Rep}_K(\mathcal{S})$
1 : / initialise $k \times m$ (fp,cnt) 2-d array
2 : **for** $i \in [k]$
3 : $\quad A[i] \leftarrow [(\star, 0)] \times m$
4 : **for** $x \in \mathcal{S}$
5 : $\quad A \leftarrow \textsc{Up}_K(A, \text{up}_x)$
6 : **return** A

$\textsc{Qry}_K(A, \text{qry}_x)$
1 : $(p_1, \ldots, p_k) \leftarrow R(K, x)$
2 : $\text{fp}_x \leftarrow T(K, x)$
3 : $\text{cnt}_x \leftarrow 0$
4 : **for** $i \in [k]$
5 : $\quad$ **if** $A[i][p_i].\text{fp} = \text{fp}_x$
6 : $\quad\quad \text{cnt} \leftarrow A[i][p_i].\text{cnt}$
7 : $\quad\quad \text{cnt}_x \leftarrow \max\{\text{cnt}_x, \text{cnt}\}$
8 : **return** cnt_x

$\textsc{Up}_K(A, \text{up}_x)$
1 : $(p_1, \ldots, p_k) \leftarrow R(K, x)$
2 : $\text{fp}_x \leftarrow T(K, x)$
3 : **for** $i \in [k]$
4 : $\quad$ **if** $A[i][p_i].\text{fp} \notin \{\text{fp}_x, \star\}$
5 : $\quad\quad r \twoheadleftarrow [0, 1)$
6 : $\quad\quad$ **if** $r \leq d^{A[i][p_i].\text{cnt}}$
7 : $\quad\quad\quad A[i][p_i].\text{cnt} \mathrel{-}= 1$
8 : $\quad$ / overtake the counter if 0
9 : $\quad$ **if** $A[i][p_i].\text{cnt} = 0$
10 : $\quad\quad A[i][p_i].\text{fp} \leftarrow \text{fp}_x$
11 : $\quad$ / increase the count if $\text{fp} = \text{fp}_x$
12 : $\quad$ **if** $A[i][p_i].\text{fp} = \text{fp}_x$
13 : $\quad\quad A[i][p_i].\text{cnt} \mathrel{+}= 1$
14 : **return** A

Figure 3: Keyed structure $\text{HK}[R, T, m, k, d]$ **supporting point-queries for any potential stream element** $x \in \mathcal{U}$ **(**qry_x**). The parameters are a function** $R : \mathcal{K} \times \mathcal{U} \rightarrow [m]^k$**, a function** $T : \mathcal{K} \times \mathcal{U} \rightarrow \{0,1\}^n$ **for some desired fingerprint length** n**, decay probability** $0 < d \leq 1$**, and integers** $m, k \geq 0$**.**

a stream is processed: if some y arrives whose fingerprint is different than that of the current owner(s), then the current (positive) value c of $A[i][j].\text{cnt}$ is decremented with probability d^{-c}. Loosely, decrementing c is akin to $A[i][j]$ "forgetting" a prior arrival of its current owner(s); with this viewpoint, the value of $A[i][j].\text{cnt}$ is the number of times that this position "remembers" seeing its current owner(s). If y causes that number to become zero, then it becomes an owner: the stored fingerprint is changed to $\text{fp}_y = T(K, y)$, and the counter is set to 1. Note that for CMS, $M[i][j]$ "remembers" the total number of elements that it observed, but nothing about *which* elements. This observation will motivate our Count-Keeper structure, later on.

The HK provides frequency estimates via point-queries. A point-query for x returns $\max\{A[i][p_i].\text{cnt} \mid A[i][p_i].\text{fp} = \text{fp}_x, i \in [k]\}$ where $(p_1, \ldots, p_k) \leftarrow R(K, x)$ and $\text{fp}_x \leftarrow T(K, x)$; if that set is empty the point query returns 0 (no position "remembers" ever having seen x.)

Yang et al.[30] do state a probabilistic guarantee on the size of estimation errors, under an assumption that each $A[i][j]$ has one and only one owner for the duration of the stream, but the statement is insufficiently precise and its proof is flawed, so we will not quote it. In the full version of this paper[1] we recover a meaningful result.

5 ATTACKS ON CMS AND HK

In the following discussion of attacks against CMS and HK in our formal model, we will implement the mappings $R : \mathcal{U} \rightarrow [m]^k$ and $T : \mathcal{K} \times \{0,1\}^* \rightarrow \{0,1\}^n$ via calls to the **Hash**-oracle. In detail, given some unambiguous encoding function $\langle \cdot, \cdot, \cdot \rangle$, for CMS we set

[1]The full version appears on the IACR's ePrint Archive, under the same title.

$R(K,x) = (\mathbf{Hash}(\langle 1,K,x\rangle), \mathbf{Hash}(\langle 2,K,x\rangle, \ldots, \mathbf{Hash}(\langle k,K,x\rangle)))$, and for HK, we set $R(K,x)[i] = \mathbf{Hash}(\langle$"cnt", $i,K,x\rangle)$ and $T(K,x) = \mathbf{Hash}(\langle$"fp", $k+1,K,x\rangle)$. Note that the traditional analysis of CMS correctness assumes that the row-wise hash functions are sampled (uniformly) from a pairwise-independent family of functions, whereas our modeling treats the row-wise hash functions as k independent random functions from $\mathcal{U} \to [m]$. This makes the adversary's task more difficult, as our attacks cannot leverage adaptivity to exploit structural characteristics of the hash functions. For the HK, the strings "cnt" and "fp" provide domain separation, and we implicitly assume that the outputs of calls to the **Hash**-oracle can be interpreted as random elements of $[m]^k$ when called with "cnt", and as random elements of the appropriate fingerprint-space,e.g., $\{0,1\}^t$ for some constant $t \geq 0$, when called with "fp".[2]

5.1 Cover Sets

Say $\hat{n}_x$ is the CMS estimate. As noted in Section 3, the estimate $\hat{n}_x = n_x + \min_{i\in[k]}\{\sum_{y\in V_x^i} n_y\}$; thus $\hat{n}_x = n_x$ if there exists an $i \in [k]$ such that $\sum_{y\in V_x^i} n_y = 0$. Since $n_y > 0$ for any $y \in V_x^i$, we can restate this as $\hat{n}_x > n_x$ if and only if $V_x^1, \ldots, V_x^k$ are all non-empty. When this is the case, the union $C = \bigcup_{i\in[k]} V_x^i$ contains a set of stream elements that "cover" the counters $M[i][p_i]$ associated to x. Since the presence of a covering C within the stream is necessary (and sufficient) for creating a frequency estimation error for the CMS, we formalize the idea of a "cover" in the following definition.

Definition 1. *Let $\mathcal{U}$ be the universe of possible stream elements. Fix $x \in \mathcal{U}$, $r \in \mathbb{Z}$, and $\mathcal{Y} \subseteq U$. Then a set $C = \{y_1, y_2, \ldots, y_t\}$ is an $(\mathcal{Y}, x, r)$*-cover *if: (1) $C \subseteq \mathcal{Y}\backslash\{x\}$, and (2)$\forall i \in [k]$ $\exists j_1, \ldots, j_r \in [t]$ such that $R(K,x)[i] = R(K,y_{j_1})[i], \ldots, R(K,x)[i] = R(K,y_{j_r})[i]$.* ♦

For the CMS, we will be interested in $\mathcal{Y} = \mathcal{U}, r = 1$, and we will shorten the notation to calling this a 1-cover (for x), or just a cover. For the HK, we will still be interested in $r = 1$, but with a different set $\mathcal{Y}$. In particular, HK has a fingerprint function $T(K,\cdot)$, and we define the set $\mathcal{FP}(K,x) = \{y \in \mathcal{U} \mid T(K,y) \neq T(K,x)\}$. In analyzing their HK structure, Yang et al. [30], rely on there being "no fingerprint collisions", to ensure that HK have only one-sided error. (In general, the HK returned estimates may over- or underestimate the true frequency.) But, no precise definition of this term is given. We define it (by negation) as follows: stream $\vec{S}$ does not satisfy the *no-fingerprint collision* (NFC) condition with respect to x (and key K) if there exists $y, z \in \vec{S}\|x$ such that $T(K,y) = T(K,z)$ *and* $\exists i$ such that $R(K,y)[i] = R(K,z)[i]$; otherwise $\vec{S}$ does satisfy the NFC condition with respect to x (and K). In other words, $\vec{S}\|x$ cannot contain distinct elements that have the same fingerprint and share a counter position. Our analysis treats the fingerprint function $T(K,\cdot)$ and position hash functions $R(K,\cdot)[i]$ as random oracles, the particular value of K will not matter, only whether or not it is publicly known. As such, explicit mention of K can be elided without loss of generality, and we shorten $\mathcal{FP}(K,x)$ to $\mathcal{FP}_x$. Further, in the random oracle model the fingerprint computation and row position computation are independent, so the probability of their conjunction is much smaller than the simple "birthday bound" event on fingerprint collisions. Anyway, for our HK analysis (Section 5.3), we will be interested in $(\mathcal{FP}_x, x, 1)$-covers, which are just $(\mathcal{U}, x, 1)$-covers under NFC condition.

When analyzing our new CK structure (Section 6), which inherits the fingerprint function from HK, we will be interested in $(\mathcal{FP}_x, x, 2)$-covers, as $r = 1$ will no longer enable attacks to drive up estimation error.

Exploring time-to-cover. Observe that even when the stream elements and the target x are independent of the internal randomness of the structure, a sufficiently long stream will almost certainly contain a cover for x. For example, for CMS, this results in $\hat{n}_x$ being an overestimate of n_x. How long the stream needs to be for this to occur is what we explore next.

Each of CMS, HK and CK use a mapping $R(K,\cdot)$ to determine the positions to which stream elements are mapped. Let L_i^r be the number of *distinct-element* evaluations of $R(K,.)$ needed to find elements covering the target's counter in the i^{th} row r times. Then L_i^r is a negative binomial random variable with success probability $p = \frac{1}{m}$ and $\Pr[L_i^r = z] = \binom{z-1}{z-r}(1-p)^{z-r}p^r$. This is because L_i^r counts the *minimal* number of evaluations needed to find r elements $y_1, \ldots, y_r$ with $R(K,y_j)[i] = p_i$. This holds for any $i \in [k]$, and all L_i^r are independent. Thus, letting $L^r = \max\{L_1^r, L_2^r, \ldots, L_k^r\}$, we have

$$\Pr[L^r \leq z] = \prod_{i=1}^k \Pr[L_i^r \leq z] = \left(p^r \sum_{t=0}^{z-r} \binom{t+r-1}{t}(1-p)^t\right)^k.$$

When $r = 1$, this simplifies to

$$\Pr[L^1 \leq z] = \left((1-q)(1+q+q^2+\cdots+q^{z-1})\right)^k = (1-q^z)^k \quad (1)$$

with $q = 1-p$. When $r = 2$ we arrive at a more complicated expression $\Pr[L^2 \leq z] = (1 - zq^{z-1} + (z-1)q^z)^k$.

One can show that $\mathbb{E}[L^1] = \sum_{z=0}^{\infty}(1-(1-q^z)^k)$; for typical values of m, we have the very good approximation $\mathbb{E}[L^1] \approx mH_k$, H_k being the k-th harmonic number.

5.2 Cover-Set Attacks on CMS

In our attack model, if the mapping $R(K,\cdot)$ is public, we may use the **Hash** oracle (only) to find a cover set for the target x "locally", i.e., the step is entirely offline. When this is not the case, we use a combination of queries to the **Up** and **Qry** oracles to signal when a cover set *exists* among the current stream of insertions; then we make additional queries to learn a subset of stream elements that yield a cover. For space reasons, we give only one cover-finding algorithm (Figure 4), while the others can be found in the full version.

Before exploring each setting, we build up some general results. Let Cover_x^r be the event that in the execution of $\mathbf{Atk}_{\Pi}^{\text{err-fe[u,v]}}(\mathcal{A})$, the adversary queries the **Up**-oracle with $\text{up}_{e_i}, \ldots, \text{up}_{e_t}$ and $e_1, \ldots, e_t$ is an r-cover for the target. For concision, define random variable $\text{Err} = \mathbf{Atk}_{\Pi}^{\text{err-fe[u,v]}}(\mathcal{A})$. We will mainly focus on $\mathbb{E}[\text{Err}]$ when analyzing the behavior of structures, so here we observe that the non-negative nature of Err allows us to write $\mathbb{E}[\text{Err}] = \sum_{\xi>1}\Pr[\text{Err} \geq \xi]$. In determining the needed probabilities, it will be beneficial to condition on Cover_x^r, as this event (for particular values of r) will be crucial for creating errors.

Our attacks against CMS (and, later, HK and CK) have two logical stages. The first stage finds the necessary type of cover

[2] This separation could be more directly handled by augmenting the attack model with an additional hashing oracle, but for simplicity and ease of reading, we chose not to do so.

for the target x, and the second stage uses the cover to drive up the estimation error. The first stage is the most interesting, as the second will typically just insert the cover as many times as possible for a given resource budget (q_H, q_U, q_Q). We note that whether or not the first stage is adaptive depends on the public/private nature of the structure's representation and hash functions, whereas the second stage will always be adaptive.

Say **Up**-query budget (i.e., number of adversarial stream elements) is fixed to q_U, and for the moment assume that the other query budgets are infinite. Let some $q'_U \leq q_U$ of the **Up**-queries be used in the first stage of the attack. The number q'_U is a random variable, call it Q, with distribution determined by the randomness of the structure and coins of the attacker. So, $\mathbb{E}[\mathsf{Err}]$ may depend on the value of Q, and then we calculate the expectation as $\mathbb{E}[\mathbb{E}[\mathsf{Err} \mid Q]]$. After a cover C is found by the first stage (so Cover^1_x holds), the second stage can insert C until the resource budget is exhausted. Note that each insertion of C will increase the CMS estimation-error by one. Our attacks ensure that $|C| \leq k$, and so the number of C-insertions in the second stage is at least $\left\lfloor \frac{q_U - Q}{k} \right\rfloor$. Letting $0 \leq T \leq q_U$ be the maximum number of **Up**-queries allowed in the first stage (i.e. $Q \leq T$), we have

$$\mathbb{E}[\mathsf{Err}] \geq \sum_{q'_U=0}^{T} \left\lfloor \frac{q_U - q'_U}{k} \right\rfloor \Pr\left[\, \mathsf{Cover}^1_x \mid Q{=}q'_U \,\right] \Pr\left[\, Q{=}q'_U \,\right].$$

Public hash and representation setting. The public hash setting allows to find a cover using the **Hash** oracle only (i.e., Q=0). This step introduces no error; $\mathbb{E}[\mathsf{Err}] = \left\lfloor \frac{q_U}{k} \right\rfloor \Pr\left[\, \mathsf{Cover}^1_x \mid Q{=}0 \,\right]$. Given our definition of L^1 as the minimal number of $R(K, \cdot)$ evaluations to find a cover, the cover-finding step of the attack requires $k(1{+}L^1)$ **Hash**-queries: k to evaluate $R(K, x)$, and then kL^1 to find a cover. Say q_H is the **Hash**-oracle budget for the attack. A cover is then found iff $L^1 \leq \frac{q_H - k}{k}$. Assuming $q_U > k$ (so that a found cover is inserted at least once) and using (1) we arrive at

$$\Pr\left[\mathsf{Cover}^1_x \mid Q{=}0\right] = \left(1 - (1{-}1/m)^{\frac{q_H}{k}-1}\right)^k \tag{2}$$

implying $\mathbb{E}[\mathsf{Err}] \geq \left\lfloor \frac{q_U}{k} \right\rfloor \left(1 - (1 - 1/m)^{\frac{q_H}{k}-1}\right)^k$. For $q_H/k \gg 1$, which is likely as q_H is *offline* work and practical k are small, $\mathbb{E}[\mathsf{Err}] \approx q_U/k$. The pseudocode of the attack can be found in the full version.

Private hash and private representation setting. This is the most challenging setting to find a cover: the privacy of hash functions effectively makes local hashing useless, and the private representation prevents the adversary from learning anything about the result of online hash computations.

Our attack (Figure 4) starts by first querying for the estimated frequency of target x followed by inserting distinct random elements ($\neq x$), and querying for the estimated frequency of x after each insertion. Call this stream of elements $\vec{I}$. This continues until the estimate on x increases by 1, which signals that a full 1-cover for x is contained within the stream $\vec{I}$ of initial insertions. Next, we extract from $\vec{I}$ a 1-cover of size $\leq k$. Call z_1 the last element inserted as a part of $\vec{I}$. As this insertion caused the estimate to increase it must be that z_1 covers at least one counter of x. Thus, we set our round-one candidate cover $C_1 \leftarrow \{z_1\}$. Next, we keep inserting the cover until the estimate for x stops changing, signalling that counters that x maps to that are minimal are *not* covered by C_1.

We then proceed as follows. In each round $i = 2, \ldots$ we first keep reinserting $\vec{I} \setminus C_{i-1}$ *in order* until some z_i increases the CMS estimate for x again, then we set $C_i \leftarrow C_{i-1} \cup \{z_i\}$. We then reinsert the cover until x's estimate stops changing signalling, again, that the minimal x-counters are not covered by C_i and allowing to find a "fresh" covering element of x in the subsequent round.

This procedure ensures that after $\ell \leq k$ rounds a full cover is found. Each round i adds exactly one new element z_i to the incomplete cover C_{i-1}, and there are only k counters to cover. Let d_i be the number of re-insertions of C_i in round i. Then, the number of **Up**-queries required to reach C_ℓ is $q'_U \leq \ell|\vec{I}| + \sum_{i=1}^{\ell-1} id_i$. So, C_ℓ can be potentially reinserted $\lfloor (q_U - q'_U)/\ell \rfloor$ times, each time increasing the error on x.

From the attack's construction, it is easy to see that $\delta_i = d_i - 1$ re-insertions of C_i each increase the error by 1, and that, additionally, the error increases by 1 at each detection of a fresh z_i. Assuming q_Q is not the limiting factor and replacing $|\vec{I}|$ with L_1 (as $\mathbb{E}[|\vec{I}|] = L_1$), and simplifying, we get

$\mathrm{CoverAttack}^{\mathbf{Up},\mathbf{Qry}}(x, \perp, \perp)$

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

$\mathrm{Uncover}^{\mathbf{Up},\mathbf{Qry}}(x, a', \mathrm{cover})$

```
1 : b' ← −1
2 : while a' ≠ b'
3 :     if (q_U − |cover| + 1)Up-
4 :         or q_Q Qry-queries:
5 :             return cover
6 :     b' ← a'
7 :     for y ∈ cover : Up(y)
8 :     a' ← Qry(x)
9 : return a'
```

$\mathrm{FindCover}^{\mathbf{Up},\mathbf{Qry}}(x)$

```
1 : / find 1-cover for x
2 : cover ← ∅
3 : found ← False
4 : I⃗ ← ∅; a ← Qry(x)
5 : while not found
6 :     if q_U Up- or q_Q Qry-queries:
7 :         return cover
8 :     y ↞ U \ (I⃗ ∪ {x})
9 :     I⃗ ← I⃗ ∪ {y}
10 :    Up(y); a' ← Qry(x)
11 :    if a' ≠ a :
12 :        cover ← {y}
13 :        found ← True
14 : for i ∈ [2, 3, ..., k]
15 :    a ← Uncover^{Up,Qry}(x, a', cover)
16 :    if a = cover : return cover
17 :    for y ∈ I / in order of insertion to I
18 :        if q_U Up- or q_Q Qry-queries:
19 :            return cover
20 :        Up(y); a' ← Qry(x)
21 :        if a' ≠ a :
22 :            cover ← cover ∪ {y}
23 :            I⃗ ← I \ {y}
24 :            break
25 : return cover
```

Figure 4: Cover Set Attack for the CMS in private hash function and private representation setting. The attack is parametrised with the update and query query budget q_U and q_Q.

$\mathsf{Err} \geq \left\lfloor \left(\frac{\ell+1}{2} + \frac{1}{\ell} \left(q_U + \sum_{i=1}^{\ell-1} (\ell - i)\delta_i \right) - L^1 \right) \right\rfloor$. We note that Err is a function of several random variables: $L^1, \ell, \{\delta_i\}_{i \in [\ell-1]}$. For practical values of k, m (e.g., $k = 4$, with $m \gg k$) it is likely that $\ell = k$; $\ell < k$ if and only if at least one $z \in C$ cover multiple x-counters which is unlikely for small $k \ll m$. So, we approximate Err with $\widehat{\mathsf{Err}}$ by replacing ℓ with k, dropping the floor operation, and arrive at $\mathbb{E}[\widehat{\mathsf{Err}}] \approx (k+1)/2 + 1/k \left(q_U + \sum_{i=1}^{k-1} (k-i)\mathbb{E}[\delta_i] \right) - \mathbb{E}[L^1]$. Rearranging and using the very tight approximation $\mathbb{E}[L^1] \approx mH_k$: $\mathbb{E}[\widehat{\mathsf{Err}}] \approx \left(\frac{q_U}{k} - mH_k \right) + \frac{k+1}{2} + \left(\frac{1}{k} \sum_{i=1}^{k-1} (k-i)\mathbb{E}[\delta_i] \right)$. We expect $\mathbb{E}[\delta_i]$ to be upper bounded by a constant that is small relative to $m, q_U/k$, and thus, the dominant term in $\mathbb{E}[\widehat{\mathsf{Err}}] \approx \mathbb{E}[\mathsf{Err}]$ will be $\frac{q_U}{k} - mH_k$. This is observed experimentally (see Table 2). We give more details about this attack (including considering the case in which q_Q is the limiting adversarial resource) in the full version.

Public hash and private representation setting. Observe that the public representation is never used in our attack in the public hash and public representation setting. Therefore, in this public hash and private representation setting, the same attack can be used. The same analysis applies.

Private hash and public representation setting. The public representation allows for an attack similar to our attack in the public hash settings. Here, we use the **Up**-oracle instead of the **Hash**-oracle to find a cover. By comparing the state before and after adding an element it is easy to deduce the element's counters (as they are the only ones to change). Our attack first adds the target to get its counters. Then, we keep inserting *distinct* elements, comparing the state before and after until a cover C is found. By the definition of L^1, the cover is found with $(q'_U = 1 + L^1)$ **Up**-queries, and is after reinserted $\lfloor (q_U - q'_U)/|C| \rfloor$ times, each time adding one to the estimation error. Hence, $\mathsf{Err} \geq \lfloor (q_U - 1 - L^1)/|C| \rfloor \geq \lfloor (q_U - 1 - L^1)/k \rfloor$ and $\mathbb{E}[\mathsf{Err}] \geq \frac{q_U - 1 - \mathbb{E}[L^1]}{k} \approx \frac{q_U - mH_k}{k}$.

5.3 Cover-Set Attacks on HK

By examining the HK pseudocode, it is not hard to see that when a stream $\vec{S}$ satisfying the NFC condition is inserted in the HK structure, over-estimations are not possible; any error in frequency estimates is due to underestimation. We also note that if $\vec{S}$ satisfies the NFC condition, then any cover that it contains for $x \in \mathcal{U}$ must be a $(\mathcal{FP}_x, x, r)$-cover. In attacking HK, we will build $(\mathcal{FP}_x, x, 1)$-covers; as such, in this section we will often just say "cover" as shorthand.

The intuition for our HK-attacks is, loosely, as follows. If one repeatedly inserts a cover for x, *before x is inserted*, then the counters associated to x will be owned by members of the cover, and the counter values can be made large enough to prevent any subsequent appearances of x from decrementing these counters with overwhelming probability. We will sometimes say that such hard-to-decrement counters are "locked-down". As such, the HK estimate $\hat{n}_x$ will be zero, even if $n_x \gg 0$.

We note that attacks of this nature would be particularly damaging in instances where the underlying application uses HK to identify the most frequent elements in a stream $\vec{S}$. With relatively few insertions of the cover set, one would be able to hide many occurrences of x. DDoS detection systems, for example, rely on compact frequency estimators to identify communication end-points that are subject to an abnormally large number of incoming connections [19]. In this case, the target x is an end-point identifier (e.g., an IP address and/or TCP port). Being able to hide the fact that the end-point x is a "heavy hitter" in the stream of incoming flow destinations could result in x being DDoSed.

Interestingly, while a cover is necessary to cause a frequency estimation error for x, it is *not* sufficient. Unlike the CMS, whose counters are agnostic of the order of elements in the stream, the HK counters have a strong dependence on order. Thus, if x is a frequent element and many of its appearances are at the beginning of the stream, then *it* can lock-down its counters; a cover set attack is still possible, but now the number of times the cover must be inserted may be much larger than the frequency (so far) of x.

Setting the attack parameter t. Say our attack's resource budget is (q_H, q_U, q_Q). The HK attacks find a cover $C = \{z_1, z_2 \dots\}$ and then insert it t times. We set the value t such the the probability p of decrementing the any of the target's counters with subsequent insertions of x is sufficiently small. For our experiments we set $p = 2^{-128}$.

Let D_i^t be the event that at the end of the attack $A[i][p_i].\mathrm{fp} = \mathrm{fp}_x$ given that at some point during the attack we had $A[i][p_i].\mathrm{cnt} = t$ with $A[i][p_i].\mathrm{fp} = \mathrm{fp}_{z_i}, z_i \neq x$. Let $\left(D^t\right) = \bigvee_{i=1} D_i^t$. Then, $\Pr[D_i^t] \leq \binom{q_U}{t} \prod_{j=1}^{t} d^j \leq (q_U)^t \, d^{\frac{t(t+1)}{2}}$.

Say $f(t) = k\,(q_U)^t \, d^{\frac{t(t+1)}{2}}$. If the attack set $A[i][p_i].\mathrm{cnt} = t$ with $A[i][p_i].\mathrm{fp} = \mathrm{fp}_{z_i}, z_i \neq x$ for each i, then the probability of x overtaking any of its counters by the end of the attack is bounded by $\Pr[\bigvee_{i=1}^{k} D_i^t] \leq f(t)$. This motivates the selection of t in our attacks as the smallest $t \geq 1$ such that $f(t) \leq p$.

Public hash and public representation setting. This attack is similar to the CMS attack for the public hash setting, but with a few tweaks. The cover is inserted only t times and then the **Up** budget is exhausted by inserting target x (at least $(q_U - tk)$ times) to accumulate error. If $\neg D^t$ then this process introduces the error of at least $(q_U - tk)$. Thus, as the cover finding step uses **Hash** only and induces no error,

$$\mathbb{E}\left[\mathsf{Err}\right] \geq (q_U - tk)(1-p) \Pr\left[\mathsf{Cover}_x^1 \mid Q = 0\right].$$

For the term $\Pr\left[\mathsf{Cover}_x^1 \mid Q = 0\right]$ we can simply apply the same bound as for the CMS attack (Equation (2)) obtaining

$$\mathbb{E}\left[\mathsf{Err}\right] \geq (q_U - tk)(1-p)\left(1 - (1 - 1/m)^{\frac{q_H}{k} - 1}\right)^k.$$

The pseudocode of the attack can be found in the full version.

Private hash and private representation setting. The attack for this setting starts by inserting x once. Starting with an empty HK implies that then x owns all of its buckets, i.e., $A[i][p_i].\mathrm{fp} = \mathrm{fp}_x$ for all rows i, with their associated counters $c_1, \dots, c_k$ set to one, setting x's current frequency estimate $a = \max_{i \in [k]} \{c_i\} = 1$. The attack then keeps inserting *distinct* elements until the frequency estimate for x drops to 0, i.e, $A[i][p_i].\mathrm{fp} \neq \mathrm{fp}_x$ for all rows i. Let $\mathcal{I}_1$ be the set of inserted elements $\neq x$ at the moment that this happens, and the last inserted element was z_1. Then, z_1 must share at least one counter with x (the one that changed $A[i][p_i].\mathrm{fp}$ from fp_x most

recently). So, we set our round-one candidate cover set $C_1 \leftarrow \{z_1\}$ and insert t times to the HK. Now we are at the point when all $c_1, \ldots, c_k$ are owned by elements $\neq x$, and all but one are of value one. Note that inserting $\mathcal{I}_1$ increased the estimate error by one.

The adaptive portion of our attack proceeds as follows. In each round $i = 2, \ldots$ we first keep reinserting x until $HK(x)$ reaches 1. Let d_i be the number of these reinsertions. Hence, these reinsertions increased the estimate error by $d_i - 1$. At this point, at least one counter $c_1, \ldots, c_k$ is owned by x and all counters owned by x are set to 1. Then, we search for a new element to create our round-i cover set candidate C_i, by inserting *new distinct* elements, until we find a z_i that drops $HK(x)$ to 0. We set $C_i \leftarrow C_{i-1} \cup \{z_i\}$ and insert z_i t times. At this point, all counters $c_1, \ldots, c_k$ are owned by elements $\neq x$ again, and all are of value one, but the ones covered by C_i which (very likely) hold a value strictly greater than 1 and (very) close to t.

The procedure ensures that after some $\ell \leq k$ rounds we have found a complete 1-cover with (very) high probability. Each round i adds maximally one new element to the incomplete cover C_{i-1}. The added element covers whatever x is owning at the beginning of the round. Thus, with (very) high probability, counters owned by x in the round are not covered by C_{i-1}. This is because all the counters covered by C_{i-1} were set to value t (or a value close to t with very high probability[3]) at some point, and the selection of t makes the probability of later overtaking one such counter (very) small. There are only k counters to cover and so with (very high) probability having only k rounds suffices to find a cover.

Let $\mathcal{I}_i$ be the set of inserted elements $\neq x$ in each round. We get the number of **Up**-queries required to complete k rounds is $q'_U \leq \sum_{i=1}^{k}(d_i + |\mathcal{I}_i| + t) = \sum_{i=1}^{k}(d_i + |\mathcal{I}_i|) + tk$.

So, x can be potentially inserted $q_U - q'_U$ times, accumulating some additional error[4]. Say C is the attack's maximal round candidate cover. Whenever $\neg\left(D^t\right)$, adding z_i t times to the HK incremented one of the x's counters, not yet set to value t by elements in C_{i-1}. If, in addition, we have $|C| = k$, k different elements set k different counters of x (i.e. all of the x's counters) to t making them impossible to decrement later.

Therefore, after the rounds to reach C are completed every further insertion of x ($q_U - q'_U$ of them) increased the error by 1. Note that $|C| = k$ implies the attack completed exactly k rounds and $\left[\mathsf{Err} \mid \neg\left(D^t\right), |C| = k\right] \geq q_U - \sum_{i=1}^{k}\left[|\mathcal{I}_i| \mid \neg\left(D^t\right), |C| = k\right] - tk$. Let D_i be the set of rows i with $A[j][p_j].\mathrm{fp} = \mathrm{fp}_x$ (i.e. x owning the counter) after the reinsertion step. Say $c_{i,j}$ are the values of $A[j][p_j].\mathrm{cnt}$ after the i-th round reinsertion step. Say $Y_{i,j}$ counts the minimal number of distinct element insertions to overtake the counter in row $j \in D_i$ from x after the i-th round reinsertion step, i.e., the minimal number of distinct evaluations of $R(K,\cdot)$ to set $A[j][p_j].\mathrm{fp} \neq \mathrm{fp}_x$. Then, $Y_{i,j}$ is a geometric random variable with $p = \frac{d^{c_{i,j}}}{m}$, $d^{c_{i,j}}$ coming from the probabilistic decay mechanism and $|\mathcal{I}_i| = \max_{j \in D_i}\left\{Y_{i,j}\right\}$. Moreover, $c_{i,j} = 1$ for all $j \in D_i$ – that is counters owned by x always have value 1. As $|D_1| = k$ we have that $|\mathcal{I}_1| = \max_{j \in D_1}\left\{Y_{1,j}\right\}$ is essentially L^1 with $p = \frac{d}{m}$. Since $|D_i| \leq k$ and all $Y_{i,j}$ are positive and i.i.d. we have that $\mathbb{E}\left[|\mathcal{I}_i|\right] \leq \mathbb{E}\left[|\mathcal{I}_1|\right]$. Hence, $\frac{m}{d}H_k \geq \mathbb{E}\left[|\mathcal{I}_i|\right]$. Skipping the derivation due to space constraints (refer to the full paper), this implies that $\mathbb{E}\left[\mathsf{Err}\right] \geq (q_U - tk)\Pr\left[\neg\left(D^t\right) \wedge |C| = k\right] - \frac{km}{d}H_k$.

We expect $\Pr\left[\neg\left(D^t\right) \wedge |C| = k\right] \approx 1$ and $\mathbb{E}\left[\mathsf{Err}\right] \approx q_U - tk - \frac{km}{d}H_k$. We confirmed this experimentally as seen in Table 2. The attack's pseudocode can be found in the full version.

Public hash and private representation setting. As with the CMS, the same attack and analysis applies from the public hash and public representation setting.

Private hash and public representation setting. The public representation allows us to design an attack similar to the attack for the public hash settings, but, as with the CMS attack in the setting, we need to find the cover using the **Up** oracle. Starting with an empty filter, the attack first inserts x, such that x is guaranteed to own all of its counters. Then, we keep adding *distinct* elements, until all the $A[i][p_i].\mathrm{fp}$ that once belonged to x has changed, in turn signaling the cover for x has been found. We give a pseudocode description of this attack in the full version.

Adding any $y \neq x$ has $\frac{d}{m}$ probability to change $A[i][p_i].\mathrm{fp}$ after the single initial insertions of x. Let Y_i be the minimal number of distinct-element $\neq x$ insertions before $A[i][p_i].\mathrm{fp}$ changes from fp_x. We observe that Y_i is a geometric random variable with success probability $p = \frac{d}{m}$. Set $Y = \max_{i \in [k]}\{Y_i\}$. So, our cover finding step requires ($q'_U = 1 + Y$) **Up**-queries to complete - 1 query to insert x, and then Y to find a cover. Say q_U is the total **Up**-query budget. After the cover finding step, we insert cover C t times to lock-down the counters followed by $q_U - q'_U - t|C|$ insertions of x. Each x-insertion added one to the error if $\neg\left(D^t\right)$ and

$$\begin{aligned}\mathbb{E}[\mathsf{Err}] &\geq \mathbb{E}\left[\mathsf{Err} \mid \neg\left(D^t\right)\right]\Pr\left[\neg\left(D^t\right)\right] \\ &\geq (q_U - 1 - \mathbb{E}\left[Y\right] - tk)\Pr\left[\neg\left(D^t\right)\right] \\ &\approx q_U - \frac{m}{d}H_k - tk.\end{aligned}$$

The last approximation comes from assuming t is set such that $\Pr\left[\neg\left(D^t\right)\right] \approx 1$, and observing that Y is essentially L^1 with $p = \frac{d}{m}$ (i.e. m replaced with $\frac{m}{d}$).

6 COUNT-KEEPER

In Figure 5 we present the Count-Keeper (CK) data structure. At a high level, CK uses information from both CMS and HK (with $d = 1$) to create frequency estimates that are more accurate than either CMS or HK (alone) when the stream is "honest", and that are more robust in the presence of adversarial streams. After describing the structure, we will provide analytical support for its design, i.e., why it is more accurate and robust. To summarize this very briefly and informally: CK is more accurate because its HK component can decrease the effect of "collision noise" that drives up the values held at the relevant $M[i][p_i]$ in the CMS component; and it is more robust because a 1-cover no longer suffices to create estimation errors (minimally, a 2-cover is needed) and, unlike either CMS or HK alone, CK can detect when the state of M, A is "abnormal" and prone to producing spurious estimates.

[3]We could have z_i simultaneously covering more not yet covered counters. Then, adding z_i t times fixes one counter to t, and the others to t with at probability at least 0.9 – the other counters might have been owned by some others elements but are definitely of value one, so each of them gets "taken" by z_i in the first insertion with probability 0.9.

[4]We say potentially as the **Qry**-query budget might be a limiting factor.

6.1 Structure

At initialization, the CK initializes a standard CMS (initialized in the structure as M) and a HK with the decay parameter $d = 1$ (initialized in the structure as A) in their usual way. We set the substructures to be of the same number of rows and buckets and let the elements hash to the same counters' positions in each substructure using the same row hash functions.

To insert a stream element x arrives, we run the CMS and HK update procedures $M \leftarrow \text{UP}_K^{\text{CMS}}(M, \text{up}_x)$ and $A \leftarrow \text{UP}_K^{\text{HK}}(M, \text{up}_x)$, respectively. We note that the same positions $(p_1, \ldots, p_k) \leftarrow R(K, x)$ are visited in both procedures; thus the same elements are observed by $M[i][p_i]$ and $A[i][p_i]$. By "observed", we mean that both $M[i][p_i]$ and $A[i][p_i]$ maintain summary information about the same substream, namely the substream of elements z such that $p_i = R(K, z)[i]$.

When queried for the frequency estimate of an element $x \in \mathcal{U}$, CK first computes the CMS and HK estimates, which we will write as CMS(x) and HK(x) for brevity. If CMS(x)=HK(x), then we return their shared response. We will see precisely why this is the correct thing to do, but loosely, it is because (under the no fingerprint collision assumption) $\text{HK}(x) \le n_x \le \text{CMS}(x)$. If $\text{CMS}(x) \neq \text{HK}(x)$ then CK proceeds row-by-row, using the information held at $A[i][p_i]$ to refine the summary information held at $M[i][p_i]$. If any of the $A[i][p_i]$.fp are uninitialized, then we are certain that $n_x = 0$; had *any* stream element been mapped to this position, the fingerprint would no longer be uninitialized.In this case, CK(x) returns 0.

Now assume that none of the $A[i][p_i]$ have uninitialized fingerprints, and $\text{CMS}(x) \neq \text{HK}(x)$. To explain our row-by-row refinements, let us define two sets $I_x = \{i \in [k] \mid A[i][p_i].\text{fp} = \text{fp}_x\}$ and $\hat{I}_x = \{i \in [k] \mid A[i][p_i].\text{fp} \neq \text{fp}_x\}$, i.e., the subset of rows in M (and A) that are "owned" and not "owned" (resp.) by x. Observe that we can write the CMS estimate for x as

$$\text{CMS}(x) = \min\left\{\min_{i \in I_x}\{M[i][p_i]\}, \min_{i \in \hat{I}_x}\{M[i][p_i]\}\right\}$$

so for each row $i \in [k]$, we have two cases to consider. For each case, CK maintains an internal estimator: when $i \in \hat{I}_x$ the estimator is Θ_1^i, and when $i \in I_x$ the estimator is Θ_2^i. We will talk about each of these, next. The upshot of this discussion is that CK defines $\Theta_1 = \min_{i \in \hat{I}_x}\{\Theta_1^i\}$, $\Theta_2 = \min_{i \in I_x}\{\Theta_2^i\}$, and its return value $\lfloor \min\{\Theta_1, \Theta_2\} \rfloor$ is always at least as good as CMS(x). Later we will argue that the CK estimate can be significantly more precise than the CMS estimate.

6.2 Correcting CMS and Correctness of CK

In what follows, we will assume the NFC condition. For sufficiently large fingerprints (e.g., τ-bit fingerprints where 2^τ is much larger than the number of distinct elements in the stream) this is reasonable. Under this assumption, CK may only overestimate n_x.

Correcting $M[i][p_i]$ when x does not "own" $A[i][p_i]$. By its design as a count-all structure, the value of $M[i][p_i] = n_x + \sum_{y \in V_x^i} n_y$. When $i \in \hat{I}_x$, we claim that $n_x \le \sum_{y \in V_x^i} n_y$. To see this, observe that if $n_x > \sum_{y \in V_x^i} n_y$ then x *would* own $A[i][p_i]$: we can pair up appearances of x with appearances of elements in $y \in V_x^i$, and

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

1: $M \leftarrow \text{zeros}(k, m)$
2: **for** $i \in [k]$
3: $\quad A[i] \leftarrow [(\star, 0)] \times m$
4: $\text{repr} \leftarrow \langle M, A \rangle$
5: **for** $x \in \mathcal{S}$
6: $\quad \text{repr} \twoheadleftarrow \text{UP}_K(\text{repr}, \text{up}_x)$
7: **return** repr

$\text{UP}_K(\text{repr}, \text{up}_x)$

1: $\langle M, A \rangle \leftarrow \text{repr}$
2: $M \twoheadleftarrow \text{UP}_K^{\text{CMS}}(M, \text{up}_x)$
3: $A \twoheadleftarrow \text{UP}_K^{\text{HK}}(A, \text{up}_x)$
4: **return** repr$\leftarrow \langle M, A \rangle$

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

1: $\langle M, A \rangle \leftarrow \text{repr}$
2: $(p_1, \ldots, p_k) \leftarrow R(K, x)$
3: $\text{fp}_x \leftarrow T(K, x)$
4: $\Theta_1, \Theta_2 \leftarrow \infty$, $\Delta \leftarrow \infty$
5: $N \leftarrow \sum_{j=1}^{m} M[1][j]$
6: $\text{flag} \leftarrow \text{False}$
7: / CMS only overestimates
8: $\text{cnt}_{\text{UB},x} \leftarrow \text{QRY}_K^{\text{CMS}}(M, \text{qry}_x)$
9: / HK only underestimates
10: $\text{cnt}_{\text{LB},x} \leftarrow \text{QRY}_K^{\text{HK}}(A, \text{qry}_x)$
11: / if upperbound equal to lowerbound
12: **if** $\text{cnt}_{\text{UB},x} = \text{cnt}_{\text{LB},x}$
13: $\quad$ **return** $\text{cnt}_{\text{UB},x}$, flag
14: **for** $i \in [k]$
15: $\quad$ / if never observed
16: $\quad$ **if** $A[i][p_i].\text{fp} = \star$
17: $\quad\quad \text{cnt}_{\text{UB},x} \leftarrow 0$
18: $\quad\quad$ **return** 0 , flag
19: $\quad$ / upper bound adjustment
20: $\quad$ / x does not own counter
21: $\quad$ **else if** $A[i][p_i].\text{fp} \neq \text{fp}_x$
22: $\quad\quad \Theta \leftarrow \frac{M[i][p_i] - A[i][p_i].\text{cnt} + 1}{2}$
23: $\quad\quad \Theta_1 \leftarrow \min\{\Theta_1, \Theta\}$
24: $\quad\quad \hat{\Delta} \leftarrow \Theta$
25: $\quad$ / x owns counter
26: $\quad$ **else if** $A[i][p_i].\text{fp} = \text{fp}_x$
27: $\quad\quad \Theta \leftarrow \frac{M[i][p_i] + A[i][p_i].\text{cnt}}{2}$
28: $\quad\quad \Theta_2 \leftarrow \min\{\Theta_2, \Theta\}$
29: $\quad\quad \hat{\Delta} \leftarrow \frac{M[i][p_i] - A[i][p_i].\text{cnt}}{2}$
30: $\quad \Delta \leftarrow \min\{\Delta, \hat{\Delta}\}$
31: $\text{cnt}_{\text{UB},x} \leftarrow \lfloor \min\{\Theta_1, \Theta_2\} \rfloor$
32: if $\Delta \ge \psi N$: $\text{flag} \leftarrow \text{True}$
33: **return** $\text{cnt}_{\text{UB},x}$, flag

Figure 5: Keyed structure $\text{CK}[R, T, m, k$, $\psi]$ supporting point-queries for any potential stream element x (qry_x). $\text{QRY}_K^{\text{CMS}}, \text{UP}_K^{\text{CMS}}$, resp. $\text{QRY}_K^{\text{HK}}, \text{UP}_K^{\text{HK}}$, denote query and update algorithms of keyed structure $\text{CMS}[R, T, m, k]$ (Figure 2), resp. $\text{HK}[R, T, m, k, 1]$ (Figure 3, but note $d = 1$). The parameters are a function $R : \mathcal{K} \times \{0,1\}^* \to [m]^k$, a function $T : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$ for some desired fingerprint length n, and integers $m, k \ge 0$. A concrete scheme is given by a particular choice of parameters. Additional flag parameter $\psi \in (0,1)$ allows the structure to raise a flag on "bad" frequency estimation (see Section 6.6).

because no element of V_x^i has the same fingerprint as x, each pair (x, y) effectively contributes 0 to the value of $A[i][p_i]$.cnt. So if $n_x > \sum_{y \in V_x^i} n_y$, the fingerprint held at $A[i][p_i]$ would be fp_x. Note that if $n_x = \sum_{y \in V_x^i} n_y$ and $i \in \hat{I}_x$, then $A[i][p_i]$.cnt = 1 and

some $y \neq x$ was the last insertion. Thus, $A[i][p_i] - 1$ is a lower-bound on the difference $\sum_{y \in V_x^i} n_y - n_x$, i.e., the number of occurrences of $y \in V_x^i$ that are not canceled out by an occurrence of x. Thus, $n_x + A[i][p_i] - 1 \leq \sum_{y \in V_x^i} n_y$, which implies that $M[i][p_i] = n_x + \sum_{y \in V_x^i} n_y \leq 2n_x + A[i][p_i] - 1$. This argument gives a proof sketch of the following lemma, whose full proof (along with full proofs for Lemma 2, Corollary 1,and Corollary 2) can be found in the full version of the paper.

Lemma 1. *Let $\vec{S}$ satisfy the NFC condition, and let $x \in \mathcal{U}$. Then for any $i \in \hat{I}_x$ we have $n_x \leq \frac{M[i][p_i] - A[i][p_i].\text{cnt}+1}{2} = \Theta_1^i$.* ♦

As this lemma holds for every $i \in \hat{I}_x$, we conclude that $n_x \leq \Theta_1 = \min_{i \in \hat{I}_x}\{\Theta_1^i\} \leq \min_{i \in \hat{I}_x}\{M[i][p_i]\}$.

Correcting $M[i][p_i]$ when x does "own" $A[i][p_i]$. Now, say that row $i \in I_x$. Under the NFC condition $A[i][p_i].\text{cnt}$ stores the number of occurrences of x that are *not* canceled out by occurrences of $y \in V_x^i$. So, we must have had at least $\sum_{y \in V_x^i} n_y \geq n_x - A[i][p_i].\text{cnt}$ occurrences of $y \in V_x^i$. This implies $M[i][p_i] \geq 2n_x - A[i][p_i].\text{cnt}$, and, by rearranging, $n_x \leq \frac{M[i][p_i]+A[i][p_i].\text{cnt}}{2}$. This sketches a proof of the following lemma, whose full proof appears in the full version of the paper.

Lemma 2. *Let $\vec{S}$ satisfy the NFC condition, and let $x \in \mathcal{U}$. Then for any $i \in I_x$ we have $n_x \leq \frac{M[i][p_i]+A[i][p_i].\text{cnt}}{2} = \Theta_2^i$.* ♦

As this lemma holds for every $i \in I_x$, we conclude that $n_x \leq \Theta_2 = \min_{i \in I_x}\{\Theta_2^i\} \leq \min_{i \in I_x}\{M[i][p_i]\}$. Combined with the conclusion of Lemma 1, we have $n_x \leq \text{CK}(x) = \lfloor \min\{\Theta_1, \Theta_2\} \rfloor \leq \text{CMS}(x)$.

Precise estimation when some $|V_x^i| \in \{0, 1\}$. If there exists an i such that $|V_x^i| = 0$, then $M[i][p_i] = A[i][p_i] = n_x$. Hence, in this special case, both $\text{CMS}(x) = n_x$ and $\text{HK}(x) = n_x$. When this is not the case, $n_x < M[i][p_i]$ for all $i \in [k]$, so $n_x < \text{CMS}(x)$. For CK, on the other hand, if there exists a row i such that $|V_x^i| = 1$, we still have $\text{CK}(x) = n_x$. Our next result, which is a corollary of Lemmas 1 and 2, shows that either one of Θ_1^i or Θ_2^i is precisely n_x, or the smaller of the two is $n_x \pm 1/2$. Thus $\text{CK}(x) = \lfloor \min\{\Theta_1, \Theta_2\} \rfloor = n_x$. The proof can be found in the full version.

Corollary 1. *Let $i \in [k]$ be such that $|V_x^i| = 1$. If the stream satisfies the NFC condition, then*

$$i \in \hat{I}_x \Rightarrow \quad n_x = \frac{M[i][p_i] - A[i][p_i].\text{cnt}}{2} + c \text{ with } c \in \{1/2, 0\},$$

$$i \in I_x \Rightarrow \quad n_x = \frac{M[i][p_i] + A[i][p_i].\text{cnt}}{2} + c \text{ with } c \in \{-1/2, 0\}.$$ ♦

Finally, we note one more case when $\text{CK}(x) = n_x$. If one of the x's buckets holds uninitialized fingerprint, i.e. $i \in [k]$ such that $A[i][p_i].\text{fp} = \star$, then $|\hat{n}_x - n_x| = 0$. This is because 1) the HK has the property that if x maps to a position in A with an uninitialized fingerprint, then x was never inserted (i.e., $n_x = 0$); and 2) we define CK to return $\hat{n}_x = 0$ if any of x's positions in A holds an uninitialized fingerprint.

6.3 Frequency estimate errors

In this section we extend the frequency estimation error analysis of CMS to CK. We have already seen that the CK estimate is never worse than the CMS estimate; in this section, we explore how much better it can be.

We begin with a simple theorem about the relationship between Θ_1 and the plain CMS estimate.

Theorem 1. *Fix an $x \in \mathcal{U}$, and let i^* be any row index such that $\text{CMS}(x) = M[i^*][p_{i^*}]$. If $i^* \in \hat{I}_x$ then either $\text{CK}(x) = n_x$, or $\left(\Theta_1 \leq \frac{\text{CMS}(x)}{2}\right)$.* ♦

Proof. If any $A[i][p_i], i \in [k]$ has an uninitialized fingerprint, then $\text{CK}(x) = n_x = 0$. Now assume this is not the case, so that $A[i][p_i].\text{cnt} \geq 1$ for all the counters associated to x. By definition $\Theta_1 = \min_{i \in \hat{I}_x} \Theta_1^i \leq \Theta_1^{i^*}$, and so $\Theta_1 \leq \frac{M[i^*][p_{i^*}] - A[i^*][p_{i^*}].\text{cnt}+1}{2} \leq \frac{\text{CMS}(x)}{2}$. □

Next, a similar theorem relates Θ_2, the plain CMS estimate, and the HK estimate (when $d = 1$).

Theorem 2. *Fix an $x \in \mathcal{U}$, and let i^* be any row index such that $\text{CMS}(x) = M[i^*][p_{i^*}]$. If $i^* \in I_x$ then either $\text{CK}(x) = n_x$ or $\left(\Theta_2 \leq \frac{\text{CMS}(x)+\text{HK}(x)}{2}\right)$.* ♦

Proof. If any $A[i][p_i], i \in [k]$ has an uninitialized fingerprint, then $\text{CK}(x) = n_x = 0$. Now assume this is not the case, so $A[i^*][p_i^*].\text{cnt} \leq \max_{i \in I_x} A[i][p_i] = \text{HK}(x)$. We have, $\Theta_2 = \min_{i \in I_x} \Theta_2^i \leq \Theta_2^{i^*} = \frac{M[i^*][p_i^*] + A[i^*][p_i^*].\text{cnt}}{2} \leq \frac{\text{CMS}(x)+\text{HK}(x)}{2}$. □

Now, if $\text{CK}(x)$ is determined by line 13 of Figure 5, then $\text{CK}(x) = \frac{\text{CMS}(x)+\text{HK}(x)}{2}$. On the other hand, if $\text{CK}(x)$ is determined by line 18, then $\text{CK}(x) = 0 \leq \frac{\text{CMS}(x)+\text{HK}(x)}{2}$. If neither of these holds, $CK(x) = \lfloor \min\{\Theta_1, \Theta_2\} \rfloor$. Thus, Theorem 1 and 2 imply $\lfloor \min\{\Theta_1, \Theta_2\} \rfloor \leq \frac{\text{CMS}(x)+\text{HK}(x)}{2}$, giving us the following lemma.

Lemma 3. *For any $x \in \mathcal{U}$, $\text{CK}(x) \leq \frac{\text{CMS}(x)+\text{HK}(x)}{2}$.* ♦

From here, it is straightforward to bound the CK estimation error, giving us the main result of this section.

Corollary 2. *Let $x \in \mathcal{U}$. If the stream satifies the NFC condition, then $\text{CK}(x) - n_x \leq \frac{\text{CMS}(x) - \text{HK}(x)}{2}$.* ♦

Consequences of Corollary 2. First, as $\text{CMS}(x)$ and $\text{HK}(x)$ approach each other — even if both are large numbers (e.g. when the stream is long and x is relatively frequent) — the error in $\text{CK}(x)$ approaches *zero*.

Next, because CMS is a count-all structure, the *worst case* guarantee is that the error $\text{CK}(x) - n_x \leq \text{CMS}(x)/2$, i.e., when $\text{HK}(x) = 0$. This occurs iff x does not own any of its counters, which implies that x is not the majority element in *any* of the substreams observed by the positions $A[i][p_i].\text{cnt}$ to which x maps. As $M[i][p_i]$ observes the same substream as $A[i][p_i]$, and $\text{CMS}(x) = \min_{i \in [k]}\{M[i][p_i]\}$, for practical values of k, m it is unlikely that all k of the V_x^i have unexpectedly large numbers of elements. Moreover, for typical distributions seen in practice (e.g., power-law distributions that have few true heavy elements), it is even less likely that *all* of the V_x^i contain a heavy hitter. Thus under "honest" conditions, we do not expect $\text{CMS}(x)$ be very large when $\text{HK}(x)$ is very small.

This last observation surfaces something that CK can provide, and neither CMS nor HK can: the ability to signal when the incoming stream is atypical. We explore this in detail in Section 6.6.

6.4 Experimental Results

We will now compare non-adversarial performance of the compact frequency estimators (CFEs) by measuring the ability of these structures in identifying the most frequent (heavy) elements of a stream. Finding the heavy elements of a stream is the typical use case of CFEs and as such these structures are used for that purpose in many systems level applications [5, 7, 10, 20, 21, 24, 30]. The ability to accurately identify these heavy elements is based on a CFE's ability to accurately make frequency estimations on these heavy elements, while maintaining the ability to make accurate frequency estimations on the non-heavy elements, such that one would be able to distinguish between the two classes of elements. Therefore, we experimentally measure the non-adversarial performance of these structure by comparing a number of performance metrics in identifying heavy elements across three different streams.

Data Streams. We have three different streams we experiment with. We sourced two streams from a frequent item mining dataset repository[5]. We also sourced an additional stream by processing a large English language novel from Project Gutenberg.

We summarize each of these three streams and why they are of particular interest to experiment on below.

(1) **Kosarak Stream:** This data collection contained anonymized click-rate data collected from visits to an online Hungarian news site. The resultant stream is of total length $8,019,015$ with $41,270$ distinct elements. As aforementioned, we sourced this stream from a frequent item mining dataset repository which is a collection of data sets meant to test frequent item finding algorithms on – the very task which we are doing. We flattened the raw collection of data such that it would resemble a stream that could be processed item-by-item.

(2) **Novel Stream:** We created a stream by processing the individual words sequentially of The Project Gutenberg eBook plaintext edition of the 1851 English-language novel *Moby-Dick; or, The Whale* by Herman Melville (ignoring capitalization and non-alphabetical characters) [23]. Long bodies of natural language obey an approximate Zipf distribution as the frequency of any word is inversely proportional to its rank in an ordered frequency list [3]. It is of interest to measure compact frequency estimators performance against data following a Zipf distribution [5, 7, 11, 21, 24, 30]. The stream is of total length $2,174,111$ with $19,215$ distinct elements.

(3) **Retail Stream:** This data collection contained anonymized shopping data from a Belgian retail store. The resultant stream is of total length $908,576$ and contains $16,740$ distinct elements. This data set is also from the frequent items mining dataset repository. As with the Kosarak stream we flattened the raw data such that it would resemble a stream after processing.

Measures and Metrics. We want to measure the performance of the CFEs of interest in the non-adversarial setting by determining how well they are able to identify and characterize the heavy elements in the streams above.

[5]http://fimi.uantwerpen.be/data/

This problem, with varying but related definitions, is referred to in the literature as the heavy-hitters problem, the hot-items problem, or the top-K problem.

The simplest of these definitions to apply is that of the top-K problem, which is to simply report the set of elements with the K highest frequencies (for some K) for a given stream. That is given elements of a stream $\vec{S} \subseteq \{e_1, e_2, \ldots, e_M\}$ with associated frequencies $(n_{e_1}, n_{e_2}, \ldots, n_{e_M})$ we can order the elements $\{e_1^*, e_2^*, \ldots, e_M^*\}$ such that $(n_{e_1}^* \geq n_{e_2}^* \geq \ldots \geq n_{e_M}^*)$. Then for some $K \in \mathbb{Z}^+$ we output the set of elements $\{e_1^*, e_2^*, \ldots, e_K^*\}$ with the K highest frequencies $(n_{e_1}^* \geq n_{e_2}^* \geq \ldots \geq n_{e_K}^*)$.

The top-K problem can be solved exactly given space linear to that of the stream by keeping an individual counter for each distinct element in the stream. It is not possible to solve exactly with space less than linear (see [28] for a formal impossibility argument), but it is a common technique to place a small data structure such as a min-heap restricted to size K on top of a CFE and by updating this small structure on each insertion once, one is able to approximate this top-K set [20, 24, 30].

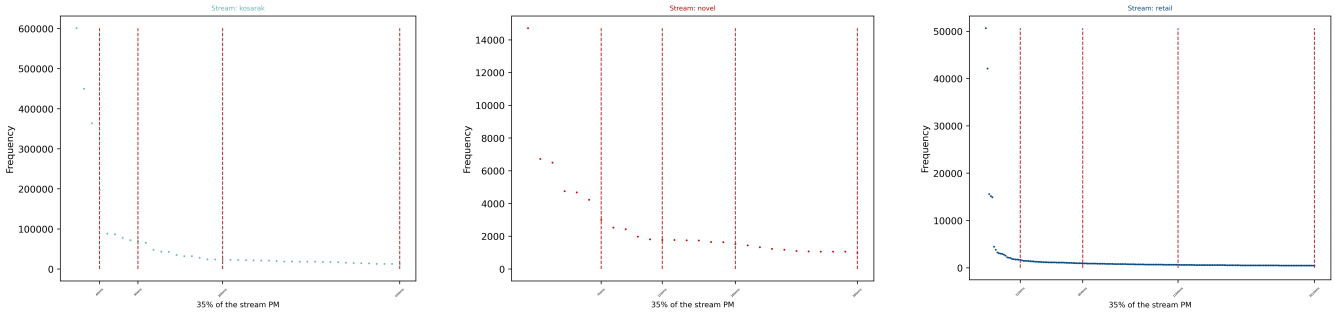
For our purposes we simply compute the approximate top-K by processing the stream with a compact frequency estimator, querying on every distinct element in the stream, and ordering elements by approximated frequency. Likewise, we compute a true top-K for each stream by processing said stream with a map linear in the size of the stream, computing a frequency for each element, and ordering by true frequency. We note that we would have achieved identical results by putting a min-heap on top of each structure with fixed sized K, updating as described in [30] and outputting its contents once the entire stream has been processed. However, for experimental purposes our approach is more extensible than the one that would be used in practice.

The number of heavy elements, or perhaps the number of heavy elements one would care about, varies depending on the stream and the application. For instance, it is noted that in a telecommunications scenario when monitoring the top outgoing call destinations of a customer typically a value of K in the range of $10 - 20$ is appropriate [17]. Moreover, when identifying the most frequent elements of interest of Zipfian distribution it is often of interest to vary K based on the parameters of the underlying distribution [7].

We select K for each stream by observing the number of clearly identifiable outliers in the underlying stream. We do this by visually inspecting the selected streams' frequency plots. We set the x-axis to enumerate all distinct elements in a stream, ordered from most to least frequent and the y-axis as those distinct elements' corresponding frequencies. We make a cut-off around the point where the frequencies went from very peaked (distinct with prominent frequency jumps from element to element) to flat (many elements with about the same frequency – the point at which the frequency differences decline less sharply). These frequency plots can be seen in Figure 6. We set $K = 20$ for the Kosarak stream, $K = 22$ for the novel stream, and $K = 22$ for the retail stream.

We measure the accuracy of the non-adversarial performance according to four different metrics.

(1) **Set Intersection Size (SIS):** This measures the size of the set intersection of the true top-K set $\mathcal{K}$ of the stream and the estimated top-K set $\tilde{\mathcal{K}}$ as reported by the CFE: SIS $= |\mathcal{K} \cap \tilde{\mathcal{K}}|$. This is measure



Kosarak stream **Novel stream** **Retail stream**

Figure 6: We plot the top 35% probability mass for each stream. That is the most frequent elements that make up 35% of the total weight of the stream (i.e. the fewest number of elements in each stream whose frequencies sum to such that when divided by the total length of the stream equal 35%). The first vertical red line in each plot is the top 20% probability mass, the second the top 25%, the third the top 30%, and the last the top 35%. From visual inspection we decided to make the top-K cut-off at, 20 for Kosarak, 22 for the Novel, and 22 for the Retail stream.

of precision on the estimated top-K set as compared to the true top-K set. A SIS of K would imply perfect precision.

(2) **Jaccard Index (JI):** The JI is a statistic that measures the similarity of two sets [27]. We use the statistic to determine the similarity of the true top-K set $\mathcal{K}$ of the stream and the estimated top-K set $\tilde{\mathcal{K}}$ as reported by the CFE. It is defined as $\text{JI} = \frac{|\mathcal{K} \cap \tilde{\mathcal{K}}|}{|\mathcal{K} \cup \tilde{\mathcal{K}}|}$. A JI can be in the range $[0, 1]$, with a JI of 1 implying a perfect characterization of the true top-K set by the CFE in its top-K estimation.

(3) **Minimal Top-$\tilde{K}$ to Capture True Top-K (MCT):** This measures determines the minimal size $L \geq K$ the estimated top-k set $\tilde{\mathcal{K}}$ would need to be to capture all elements contained in the true top-K set $\mathcal{K}$. That is if one were to order the frequency estimates of all items made by a particular CFE, we would determine the number of items one would need to examine (starting from the most-frequent going down to the least-frequent) until all the elements from $\mathcal{K}$ were contained in that ordered set. Thus, $L-K$ indicates the number of elements that fall out of $\mathcal{K}$ that are incorrectly being individually estimated to be greater than at least one element that is truly in $\mathcal{K}$.

(4) **Average Relative Error on Top-k elements (ARE)**: Average Relative Error is a standard measure to use when comparing CFEs [30]. It is defined as $\text{ARE} = \frac{1}{K} \sum_{i=1}^{K} \frac{|\hat{n}_i - n_i|}{n_i}$ where $i \in [K]$ indexes the true top-K elements for a particular stream.

Results. We crafted reference implementations for all three CFEs of interest: CMS, HK, and CK[6]. They are implemented in Python3 and use the BLAKE2b cryptographic hash function for independent row hash functions and for a fingerprint hash function in the case of CK and HK.

We are interested in comparing performance when the space used by the structures is held constant. Observe that CK is three times as large as CMS, and HK is twice as large as CMS assuming the same space is used for a counter bucket and a fingerprint bucket (in the CK and HK) across all structures. In practice these buckets could be (say) 32-bits. We picked two sets of parameters, a *standard* set and a *constrained* set to test.

[6]Source code is available at: https://github.com/smarky7CD/cfe-in-adv-envs

The standard set of parameters set $m = 2048, k = 4$ for CMS, $m = 1024, k = 4$ for HK, and $m = 910, k = 3$ for CK. This corresponds to 32.76 kB of space when using a 32-bit bucket sizes. We experimentally show that at this size all the structures are able to identify the heavy elements of the streams we test upon with minimal to no error.

The constrained set of parameters sets $m = 512, k = 4$ for CMS, $m = 256, k = 4$ for HK, and $m = 341, k = 2$ for CK. This corresponds to just 8.19 kB of space when using a 32-bit counter and fingerprint bucket sizes. In this space constrained setting the structures are still able to identify the heavy elements of the streams we test upon, but with some degree of moderate error.

For HK, we set $d = 0.9$ for all experiments, as this is the default chosen by Redis [2] and satisfies the desired properties of the exponential decay function stated in [30].

We ran 1000 trials for each structure, stream, and parameter triplet using our reference implementations. We randomize each trial on the particular choice of hash functions used for the rows (by selecting a random per-trial seed), as well as the order in which the items in the stream are processed. The latter simulates an item being randomly drawn from the underlying distribution of the stream. We averaged our four metrics for each structure, stream and parameter triplet over the 1000 trials.

We present a summary of the results in Table 1. For the *standard* parameter set we see that CK and HK perform best, being able to perfectly capture the true top-K set for each stream with their outputted estimated top-K set in *every* trial. This is indicated by the SIS and MCT being equal to K and the JI being equal to 1 for each stream. Moreover, the estimates on these top-K elements for both of these structures were very tight. The ARE over all trials and streams was ≈ 0 (ignoring a small rounding error). This indicates that CK and HK nearly perfectly individually estimated every single element in the true top-K across all trials.

CMS with the standard parameter sizing performs almost as well. Only failing to capture the true top-K set with its estimated set a few number of times over the 1000 trials. This is indicated by the SIS and MCT being very close to K and the JI being very close to 1

Structure	SIS	JI	MCT	ARE
Standard				
CK (910, 3)	20	1	20	≈ 0
	22	1	22	≈ 0
	22	1	22	≈ 0
CMS (2048, 4)	19.303	0.934	20.901	0.017
	22.999	0.999	22.001	0.009
	21.643	0.997	22.405	0.040
HK (1024, 4)	20	1	20	≈ 0
	22	1	22	≈ 0
	22	1	22	≈ 0
Constrained				
CK (341, 2)	17.189	0.757	28.695	≈ 0
	21.617	0.967	22.451	≈ 0
	13.442	0.441	209.439	0.021
CMS (512, 4)	18.241	0.841	24.567	0.125
	21.638	0.969	22.473	0.062
	18.745	0.745	41.609	0.296
HK (256, 4)	20	1	20	≈ 0
	22	1	22	0.001
	21.976	0.998	55.008	0.005

Table 1: A comparison of honest setting results between the CK, CMS, and HK compact frequency estimators. Each parameter pair (m, k) is given under the name of the corresponding structure. The statistics in each cell are listed such that they correspond to the streams in the following order: Kosarak $(K = 20)$, Novel $(K = 22)$, Retail $(K = 22)$– from ascending to descending.

for each stream. However, CMS, as it is prone to overestimation on every element, has slightly higher ARE than the other structures.

The *constrained* set of parameters presents a challenge for all the CFEs in computing individual frequency estimations on elements in the streams, and as a result computing an accurate estimated top-K. This setting only allocates CK a measly 642 individual counters to compactly represent streams that all have over 19,000 distinct elements. Under these conditions, HK performs best according to our metrics. It perfectly captures the true top-K in both the Kosarak and Novel stream, while only failing to do so in a handful of trials with the Retail stream. Moreover, the ARE is small across all streams – comparatively less than CMS with the standard parameters. HK by design prioritizes providing accurate estimates on the most frequent elements, by way of its probabilistic decay mechanism. So while it performs well on this task, it severely underestimates middling and low frequency elements at this sizing, reporting an individual frequency estimate ≈ 0 for any element that is not heavy.

CMS and CK perform less well in this small space allocation setting. While CMS performs slightly better in capturing the true top-K set within its estimated top-K set, CK continues to give better accuracy on individual point estimations of the true top-K elements across streams due to its internal sub-estimators that provide tighter estimations than CMS.

We observe in this constrained space setting across the structures measured performance is the worst on the Retail stream. This is because the Retail stream has a flatter distribution as compared to the other streams. That is to say, it has very few clearly identifiable heavy elements before containing a large collection of elements of about the same frequency. This can be seen in the frequency plot in Figure 6. The Retail element with frequency rank 22 has a true frequency of 1715 while the Retail element of frequency rank 56 has a true frequency of 1005. Comparing this to $n_{22} = 22631, n_{56} = 9559$ and $n_{22} = 1176, n_{56} = 474$, respectively for the Kosarak and Novel stream, one can see that the relative fall off in true frequency is far less pronounced within this region of the Retail stream. This in turn leads to small errors in the individual frequency estimations of elements near (but outside) the true top-K of the Retail stream propagating to the top-K estimation – by making it challenging for the CFEs to draw a clear distinction between the truly heavy elements and the nearly heavy elements. The upshot being, one needs larger structures to accurately estimate these flatter streams.

In sum, CK performs comparatively well to both CMS and HK, and in fact better than CMS when not burdened with *very* tiny space constraints. It is able to perfectly estimate the true top-K for all streams overall trials with only 2730 individual counters in the standard parameter setting, while also being adversarial robust where the others structures are not.

6.5 Attacks Against the CK

Our attacks against CK are almost one-to-one with those we present against the CMS with one major difference. Recall from Corollary 1 that if at least one counter in some row i of the element x we are querying on maps to has $|V_x^i| \leq 1$ then CK returns estimate $\hat{n}_x$ such that $\hat{n}_x = n_x$, i.e. CK(x) is a perfect estimate of x. This implies that for an error to exist in a frequency estimation of x it must be that $\forall i \in [k]$ it is necessary that $|V_x^i| \geq 2$. In the attack setting this means we need to find a 2-cover (specifically a $(\mathcal{FP}_x, x, 2)$-cover) on x to create error.

We attack CK in a two-step process, as with CMS and HK. We first find a 2-cover for our target element x and then repeatedly insert the 2-cover to create error. Under the assumption that x does not own any of its counters in the A substructure of the CK (which is guaranteed in our attack model[7]), then the Θ_1 sub-estimator will be used to make the final error evaluation **Qry** query on x. Say that after some process of finding a 2-cover for x (which will be of size $\leq 2k$ – for this discussion we will assume the size of the 2-cover is exactly $2k$) we have ω insertions to repeatedly insert the elements in the cover. Repeated and equal insertions of each of the elements in the 2-cover for x will cause the values in all of x's counters in the M substructure of the CK to be of value $\frac{\omega}{k}$. In the A substructure the value in the counters that x maps to will have value 1 and be owned by some element in the 2-cover. This is because (under the no-fingerprint collision assumption) in the initially empty structure, ownership of said counters will flip-flop on each iteration of the insertion of the 2-cover between the two distinct elements that map to these counters in accordance with the Up algorithm of the HK with $d = 1$.

Then applying the estimation from Θ_1 we see that we will generate error on x equal to $\frac{\omega}{2k}$. If we hold k constant and assume that we are attacking a CMS under the same conditions (we have found a 1-cover for a target x through some process and have ω

[7] Save for the trivial case in the public representation, private hash setting when no cover is able to be found.

insertions to accrue error) we will have an error of $\frac{\omega}{k}$, which is twice that of the CK under the same conditions. Under the same assumptions for HK, with the added one that we have already primed the cover in the structure, we will achieve an error on the target of ω, which is $\omega - \frac{\omega}{2k}$ greater than that of the CK. We will see this pattern holds when giving concrete experimental attack error results at the conclusion of this section. We present the pseudocode for all the attacks in the full version.

Public hash and representation setting. In this setting, we find a 2-cover for target x using the **Hash** oracle only, and then accumulate error for the target by repeatedly inserting the 2-cover. Each insertion of the 2-cover increases the error by one. The two cover can be inserted at least $\frac{q_U}{2k}$ as the size of the cover is $\leq 2k$. We apply the same analysis used for the CMS attack, but replace $k(1 + L^1)$ with $k(1+L^2)$ as the number of **Hash**-queries to complete the cover-finding step, as again, we now find a 2-cover. Assuming $q_U > 2k$ (so that any found C can be inserted at least once) we arrive at $\mathbb{E}[\mathsf{Err}] \geq \lfloor \frac{q_U}{2k} \rfloor \Pr\left[L^2 \leq \frac{q_H - k}{k}\right]$ Using results from Section 5.1 we can further obtain a concrete expression for $\Pr\left[L^2 \leq \frac{q_H-k}{k}\right]$.

Private hash and representation setting. Our CK attack for the setting is essentially the same as the CMS attack, except a 2-cover (as opposed to a 1-cover) is detected and repeatedly inserted to build up the error. Using analysis similar to the CMS case and assuming q_Q is not the limiting factor,

$$\mathsf{Err} \geq \left\lfloor \left(\tfrac{\ell+1}{2} + \tfrac{1}{\ell}\left(q_U + \textstyle\sum_{i=1}^{\ell-1}(\ell - i)\delta_i\right) - L^2 \right) \right\rfloor \text{ with } \ell \leq 2k$$

rounds to find a 2-cover. For reasonable sizes of the CK we mainly expect $\ell = 2k$ (for the CMS case we expected ℓ=k) and that $\mathbb{E}[\delta_1]$ are bounded by a constant that is small relative to $m, q_U/k$. Given that $k \ll m$, we expect the following to approximate $\mathbb{E}[\mathsf{Err}]$:

$$\mathbb{E}\left[\left\lfloor\left(\frac{2k+1}{2} + \frac{1}{2k}\left(q_U + \sum_{i=1}^{2k-1}(2k-i)\delta_i\right) - L^2\right)\right\rfloor\right] \approx \frac{q_u}{2k} - \mathbb{E}[L^2].$$

Public hash and private representation setting. The attack and analysis from the public hash and representation setting applies.

Private hash and public representation setting. This attack is one-to-one with the CMS attack in the same setting but again we find 2-cover as opposed to a 1-cover. Hence, $\mathbb{E}[\mathsf{Err}] \geq \frac{q_U - 1 - \mathbb{E}[L^2]}{2k} \gtrapprox \frac{q_U - 1 - 2mH_k}{2k}$.

Attack Comparisons. We implemented our attacks against all structures in all settings to experimentally verify their correctness and our analysis. In Table 2 we present a summary of results for the public hash setting (our least restrictive setting) and the private hash, private representation setting (our most restrictive setting.). We experiment on two sets of parameters, one fixing $k = 4$ and the other $k = 8$. We then select a reasonable value of m for CMS and then half it for HK and third it for CK so that the same space is used in each structure. We fix adversarial resources such that $q_H, q_U, q_Q = 2^{20}$. In practice this ensures that the number of **Hash** queries or **Qry** queries will not be the bottleneck in our attacks and that we are able to generate sufficient error in each attack to showcase overall trends. We run each attack setting and structure pairing over 100 trials, selecting a random target in each trial, and average the results.

Observe the pattern that when holding k constant and setting reasonable m values, adjusting such that CMS, CK, and HK use the same space, attacks against CK generate the least amount of error. The attacks against CK produce about half of the amount of error as opposed to the CMS attacks, and about $q_U - \frac{q_U}{2k}$ less the amount of error as opposed to the HK attacks. Moreover, observe that our analytical results closely match those of our experimental results.

6.6 Adversarial Robustness

Corollary 2 shows that the error in $\mathrm{CK}(x)$ is largest when $\mathrm{HK}(x) \ll \mathrm{CMS}(x)$. In particular, when x does not own any of its counters $\mathrm{HK}(x)$ takes on its minimal value of zero. But we can say something a bit more refined, by examining what is computed on the way to the returned value $\mathrm{CK}(x)$.

Specifically, recall that $CK(x) = \lfloor \min\{\Theta_1, \Theta_2\} \rfloor$, where Θ_1 is the smallest upperbound on n_x that we can determine by looking only at the rows that x does not own, and Θ_2 is the smallest upperbound on n_x that we can determine by looking only at the rows that x does own. Let $\Delta = |\mathrm{CK}(x) - n_x|$ be the potential error in the estimate $CK(x)$. Dropping the floor for brevity, if $\mathrm{CK}(x) = \Theta_1$ then Lemma 2 tells us that $\Delta \leq (M[i^*][p_{i^*}] - A[i^*][p_{i^*}].\mathrm{cnt} + 1)/2$, where $i^* \in \{j \mid \Theta_1^j = \min_{i \in \hat{I}_x}\{\Theta_1^i\}\}$.

Likewise, if $\mathrm{CK}(x) = \Theta_2$ then by Lemma 2 we have $n_x \leq (M[i^*][p_{i^*}] + A[i^*][p_{i^*}].\mathrm{cnt})/2$, where $i^* \in \{j \mid \Theta_2^j = \min_{i \in I_x}\{\Theta_2^i\}\}$. In this case $A[i^*][p_{i^*}].\mathrm{cnt} \leq n_x$, so we know that $\Delta \leq (M[i^*][p_{i^*}] + A[i^*][p_{i^*}].\mathrm{cnt})/2 - A[i^*][p_{i^*}].\mathrm{cnt} = (M[i^*][p_{i^*}] - A[i^*][p_{i^*}].\mathrm{cnt})/2$. Adding 1/2 to this upperbound gives the same expression as in the previous case.

Thus, we can augment the basic version of CK so that $\textsc{Qry}(\mathrm{qry}_x)$ computes Δ, and returns a boolean value flag along with the estimate of n_x. The value of flag would be set to 1 iff $\Delta \geq \psi N$, where N is the length of currently inserted stream and ψ is a parameter. We choose this condition because the non-adaptive correctness guarantees of CMS have a similar form: with k rows and m counters per row, the estimate $\mathrm{CMS}(x)$ is such that $\Pr[\,\mathrm{CMS}(x) - n_x \leq \epsilon N\,] \geq 1 - \delta$ when $\epsilon = e/m$, $\delta = e^{-k}$. From Corollary 2, in the worst case we have $\Delta \leq (1/2)\mathrm{CMS}(x) \leq (1/2)(n_x + \epsilon N)$ with probability at least $1 - \delta$.

Observe that when the frequency estimation error on an element x is large, then row i^* will be such that $M[i^*][p_{i^*}]$ will have a large value and $A[i^*][p_{i^*}].\mathrm{cnt}$ will have a value very small relative to the value in $M[i^*][p_{i^*}]$. In the worst case $A[i][p_{i^*}].\mathrm{cnt} = 1$ – in our attacks we force this to be the case. Taking $A[i^*][p_{i^*}].\mathrm{cnt} \approx 0$, observe that whether $\mathrm{CK}(x)$ is determined by Θ_1 or Θ_2, we see $\mathrm{CK}(x) \approx (1/2)M[i^*][p_{i^*}] \approx (1/2)\mathrm{CMS}(x)$ in this high error case. Then rolling in the non-adaptive CMS correctness guarantee we see $\Pr[\Delta > (1/2)(\epsilon N) - (1/2)n_x] \leq \delta$ and certainly $\Pr[\Delta > 1/2(\epsilon)N] \leq \Pr[\Delta > 1/2(\epsilon)N - (1/2)n_x]$, thus setting $\psi = (1/2)\epsilon$ (where we can derive ϵ from parameter m) can be a useful starting point for setting ψ. As a caveat, however, as N becomes large, an adversarial stream may be able to induce significant error by setting ψ in this way (due to the looseness of the CMS bound). Depending on the deployment scenario, smaller values of ψ, or even sublinear functions of N, may be more appropriate for detecting abnormal streams.

	Public Hash Setting			Private Hash, Private Rep Setting		
Structure	|cover|	Experimental Err	$\mathbb{E}$[Err]	|cover|	Experimental Err	$\mathbb{E}$[Err]
CK, ($m = 682, k = 4$)	7.96	131821.00	131072.00	7.96	130796.69	127432.90
CMS, ($m = 2048, k = 4$)	3.99	263017.82	262144.00	3.99	261116.16	257877.34
HK, ($m = 1024, k = 4$)	3.99	1047502.69	1047500.00	4.0	1038804.55	1038018.54
CK, ($m = 1365, k = 8$)	15.97	65667.10	65536.00	15.93	63776.52	56618.28
CMS, ($m = 4096, k = 8$)	8.00	131072.00	131072.00	7.99	127029.66	119939.65
HK, ($m = 2048, k = 8$)	7.96	1046434.76	1046424.00	7.98	1007439.04	996946.87

Table 2: A comparison of Err **accumulated by the different structures during attacks in the public hash setting and the private hash, private representation setting. We give the average size of the cover set and average error accumulated in each structure, setting pair over the** 100 **experiment trials. We also give the** $\mathbb{E}$[Err] **according to our analysis.**

Nonetheless, we implemented an version of CK with flag-raising (see Figure 5), and set $m = 1024, k = 4$. This corresponds to $\epsilon = 0.00265, \delta = 0.0183$. We then set $\psi = 0.0012 < \frac{1}{2}\epsilon$. Against it, we ran 100 trials of the public hash, public representation attack with $q_U = 2^{16}$, and with per-trial random target elements x. The average error was 8203.71, and in *every* trial the warning flag was raised.

For comparison, we also ran 100 trials, with the same parameters, using the non-adversarial streams from Section 6.4. In each trial, the entire stream was processed, and then we queried for the frequency of *every* element in the stream, counting the number of estimates that raised the flag. Over all 100 trials, or nearly 7.7 million estimates in total, only *three* flags were raised. These initial findings suggest that the potential for CK to flag suspicious estimates may be of significant benefit to systems employing compact frequency estimators.

ACKNOWLEDGEMENTS

Filić was supported by the Microsoft Research Swiss Joint Research Center. Markelon and Shrimpton were supported by NSF awards 1933208 and 1816375. We thank these agencies for their generous support. We also thank the reviewers for their insightful and helpful feedback.

REFERENCES

[1] Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. https://redis.io/.
[2] Redisbloom: Probabilistic data structures for redis. https://oss.redis.com/redisbloom/.
[3] Lada A. Adamic and Bernardo A. Huberman. Zipf's law and the internet. *Glottometrics*, 3(1):143–150, 2002.
[4] Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. *Journal of the ACM*, 69(2), 2022.
[5] Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J. Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Transactions on Database Systems*, 35(4), 2010.
[6] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
[7] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703, 2002.
[8] David Clayton, Christopher Patton, and Thomas Shrimpton. Probabilistic data structures in adversarial environments. In *ACM SIGSAC CCS*, 2019.
[9] Edith Cohen, Xin Lyu, Jelani Nelson, Tamás Sarlós, Moshe Shechner, and Uri Stemmer. On the robustness of countsketch to adaptive inputs. https://arxiv.org/abs/2202.13736, 2022.
[10] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
[11] Graham Cormode and Shan Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *ACM Transactions on Database Systems*, 30(1):249–278, 2005.
[12] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. In *AMC SIGCOMM*, 2000.
[13] Mia Filić, Kenny Paterson, Anupama Unnikrishnan, and Fernando Virdia. Adversarial correctness and privacy for probabilistic data structures. In *ACM SIGSAC CCS*, 2022.
[14] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *DMTCS Conference on Analysis of Algorithms*, 2007.
[15] Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In *ACM STOC*, 2013.
[16] Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. In *NeurIPS*, 2020.
[17] Nuno Homem and Joao Paulo Carvalho. Finding top-k elements in data streams. *Information Sciences*, 180(24):4958–4974, 2010.
[18] Anukool Lakhina, Mark Crovella, and Christiphe Diot. Characterization of network-wide anomalies in traffic flows. In *ACM SIGCOMM Conference on Internet Measurement*, 2004.
[19] Haiqin Liu, Yan Sun, and Min Sik Kim. Fine-grained ddos detection scheme based on bidirectional count sketch. In *International Conference on Computer Communications and Networks*, 2011.
[20] Ankush Mandal, He Jiang, Anshumali Shrivastava, and Vivek Sarkar. Topkapi: parallel and fast sketches for finding top-k frequent elements. *NeurIPS*, 2018.
[21] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *International Conference on Very Large Databases*, 2002.
[22] Luca Melis, George Danezis, and Emiliano De Cristofaro. Efficient private statistics with succinct sketches. *arXiv preprint arXiv:1508.06110*, 2015.
[23] Herman Melville. *Moby Dick; Or, The Whale*. Project Gutenberg, 1851.
[24] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems*, 31(3):1095–1133, sep 2006.
[25] Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In *Annual Cryptology Conference*, 2015.
[26] Kenneth G. Paterson and Mathilde Raynal. Hyperloglog: Exponentially bad in adversarial settings. In *EuroS&P*, 2022.
[27] Raimundo Real and Juan M Vargas. The probabilistic basis of jaccard's index of similarity. *Systematic biology*, 45(3):380–385, 1996.
[28] Tim Roughgarden and Gregory Valiant. Cs168: The modern algorithmic toolbox lecture #2: Approximate heavy hitters and the count-min sketch. page 15.
[29] Benedikt Sigurleifsson, Aravindan Anbarasu, and Karl Kangur. An overview of count-min sketch and its applications. https://easychair.org/publications/preprint/gNlw, 2019.
[30] Tong Yang, Haowei Zhang, Jinyang Li, Junzhi Gong, Steve Uhlig, Shigang Chen, and Xiaoming Li. Heavykeeper: An accurate algorithm for finding top-k elephant flows. *IEEE/ACM Transactions on Networking*, 27(5):1845–1858, 2019.