The proceedings version of this paper appears at CCS ’23. This is the full version.

Compact Frequency Estimators in Adversarial Environments

Sam A. Markelon[*], Mia Filić[†], and Thomas Shrimpton[*]

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.

[*]Florida Institute For Cybersecurity Research, University of Florida; {smarkelon,teshrim}@ufl.edu
[†]Applied Cryptography Group, ETH Zürich; mia.filic@inf.ethz.ch

Contents

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 [7] admits data-membership queries (*Does element x appear in the data?*). Bloom filters are used in applications such as increasing cache performance [23], augmenting the performance of database queries [8], indexing search results [17], and Bitcoin wallet synchronization [1]. 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 [29], Clayton, Patton and Shrimpton [10], and Filić et al. [15] — have provided detailed analyses of Bloom filters under *adaptive* attacks; the results are overwhelmingly negative. Paterson and Raynal [30] provided similar results for the HyperLogLog PDS, which can be used to count the number of distinct elements in a data collection [16].

In this work, we focus on PDS that can be used to estimate the number of times any particular element x appears in a collection of data, i.e., the *frequency* of x. Such compact frequency estimators (CFEs) are commonly used in streaming settings, to identify elements with the largest frequencies — so-called *heavy hitters* or *elephants*. Finding extreme elements is important for network planning [14], network monitoring [21], recommendation systems [26], and approximate database queries [3], to name a few applications.

The Count-min Sketch (CMS) [12] and HeavyKeeper (HK) [34] 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. [33]. The HK structure is the CFE of choice in the RedisBloom module [3], a component of the Redis database system [2].

Of particular interest to us is the 2019 ACM SIGSAC work of Clayton, Patton, and Shrimpton [10] that both furthers the adversarial analysis on Bloom filters and also presents a general model for analyzing probabilistic data structures for provable security. This paper gives a first look at the security of the Count-min sketch in adversarial environments. However, in this paper a very conservative security model for the CMS was used, which counted any overestimation of a particular element as an adversarial gain, rather than tying the security to the non-adaptive guarantees of the structure. Further, a thresholding mechanism is used to achieve security for the CMS, a solution which we deem untenable for real world uses of the CMS.

As is the case for 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).

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

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

Our attacks exploit structural commonalities of CMS and HK. At their core, each of these processes incoming data elements by mapping them to multiple positions in an array of counters, and these are updated according to simple, structure-specific rules. Similarly, when frequency estimation (or *point*) queries are made, the queried element is mapped to its associated positions, and the response is computed as a simple function of values they hold. So, our attacks concern themselves with finding *cover sets*: given a target x, find a small set of data elements (not including x) that collectively hash to all 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 $V_x^i = \{y \in \vec{S} \,|\, y \neq x \text{ and } h_i(y) = p_i\}$ 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 [34] that invalidate some of their claims about the behaviors of the HK structure. We have communicated with the authors of [34] 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 [19], the authors consider adding robustness to streaming algorithms using differential privacy. Meanwhile, Hardt and Woodruff [18], Cohen et al. [11] and Ben-Eliezer et al. [5] 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.

Preliminaries. 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. For all $m \geq 2$, let $[m] = \{1, 2, \ldots, m\}$. Let $\{0,1\}^*$ denote the set of bitstrings and let ε denote the empty string. Let $X \,\|\, Y$ denote the concatenation of bitstrings X and Y. When $\mathcal{S}$ is an abstract data-object (e.g., a (multi)set, a list) and e is an object that can be appended (in some understood fashion) to $\mathcal{S}$, we overload the $\|$ operator and write $\mathcal{S} \,\|\, e$. Let $\mathrm{Func}(\mathcal{X}, \mathcal{Y})$ denote the set of functions $f : \mathcal{X} \to \mathcal{Y}$. Let $\mathcal{A}$ and $\mathcal{B}$ be sets. We take $\mathcal{A} \cup \mathcal{B}$ to be the union of the sets, $\mathcal{A} \cap \mathcal{B}$ to be the intersection of the sets, and $\mathcal{A} \setminus \mathcal{B}$ to be set-theoretic difference of $\mathcal{A}$ and $\mathcal{B}$. 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 $\text{zeros}(k, m)$ 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. [10]. We give a terse description below; see [10] 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:

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

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 $\mathsf{repr} \twoheadleftarrow \textsc{Rep}_K(\mathcal{S}), a \leftarrow \textsc{Qry}_K(\mathsf{qry})$, and $\mathsf{repr} \twoheadleftarrow \textsc{Up}_K(\mathsf{up})$ for the execution of these algorithms.

We formalize Rep as randomized to admit defenses against offline attacks and, as we will see, per-representation randomness will play an important role in achieving our notion of correctness in the presence of adaptive adversaries. Both Rep and the Up algorithm can be viewed (informally) as mapping data objects to representations. — explicitly so in the case of Rep, and implicitly in the case of Up — so we allow Up to make per-call random choices, too.

Note that Up takes a function operating on data objects as an argument, even though Up itself operates on *representations* of data objects. This is intentional, to match the way these data structures generally operate. In a data structure representing a set or multiset, we often think of performing operations such as 'insert x' or 'delete y'. When the set or multiset is not being stored, but instead modeled via a representation, the representation must transform these operations into operations on the actual data structure it is using for storage. A Bloom filter, for example, will handle an 'insert x' query by hashing x and setting the resulting bits in the filter to 1. 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 $\mathsf{qry}_x(\mathcal{S})$ returns the number of occurrences n_x of x in data-object $\mathcal{S}$. We also note that the query algorithm Qry is deterministic. This reflects the overwhelming behavior of data structures in practice, in particular those with space-efficient representations.

For the structures we consider, the Rep algorithm can always be assumed to take an empty data object as input. This is without loss, as creating an initial representation with a non-empty data object will be equivalent to beginning with an "empty" initial representation, then making a sequence of Up operations to insert the elements of the data object.

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 $v = 0$, 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 $\mathbf{Hash}\colon \mathcal{X} \to \mathcal{Y}$, 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.

Remark. Conventionally, one would define an "advantage" function over the security experiment, and there are various interesting ways this could be done. As examples, one could parameterize by a threshold

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

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

$\mathbf{Qry}(\mathsf{qry})$
    $\mathbf{return}\ \mathrm{QRY}_K(\mathsf{repr}, \mathsf{qry})$

$\mathbf{Hash}(X)$
    $\mathbf{if}\ X \notin \mathcal{X} : \mathbf{return}\ \perp$
    $\mathbf{if}\ H[X] = \perp$
        $H[X] \twoheadleftarrow \mathcal{Y}$
    $\mathbf{return}\ H[X]$
```

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 $\mathcal{X} \rightarrow \mathcal{Y}$ (i.e., a random oracle), and is implicitly provided to REP, UP and QRY.

function $T: \mathbb{Z} \rightarrow \mathbb{Z}$, and have the advantage measure the probability that the value $|\hat{n}_x - n_x| > T(q_U)$; or, one could compare this value to known non-adaptive error guarantees. As we will not be proving the security of any structures, we use $\mathbf{Atk}^{\text{err-fe[u,v]}}_{\Pi}(\cdot)$ as a precise description of the attack setting. We will explore *lower* bounds on the values returned by the experiment, for explicit attacks that we give.

We capture various settings related to the view of the adversary in our attack interface. We have a setting in which the data structure representation is kept private from the adversary, and we also have a setting in which the specific choice of hash functions selected by a particular representation are kept private from the adversary. These settings can be examined together, separately, or both can be disregarded and the adversary can be given a "full view". That is we consider when the both the representation and hash functions are private, when the representation is public and the hash functions are private, when the representation is private and the hash functions are public, and when both the representation and hash functions are public.

In practice the private representation setting occurs due to suppression of information leaked by the oracles. In particular in this setting, the **Rep** and **Up** oracles return nothing, thus leaking nothing about the underlying data representation. Further, we make hash functions "private" by keying them with a (non-empty) randomly generated secret key.

3 Count-min Sketch

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 $(p_1, \ldots, p_k) = (h_1(x), \ldots, h_k(x))$ 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 $\mathrm{QRY}(\mathsf{qry}_x)$ returns $\hat{n}_x = \min_{i \in [k]}\{M[i][p_i]\}$. We note that (in the insertion-only model) it must be that $\hat{n}_x \geq n_x$. To see this, let $V^i_x = \{y \in \vec{S} \mid y \neq x \text{ and } R(y)[i] = p_i\}$ be the set of elements that "collide" with x's counter in the i-th row. Then we can write $M[i][p_i] = n_x + \sum_{y \in V^i_x} n_y$, where $n_y \geq 0$. Viewed

```
$\text{Rep}_K(\mathcal{S})$
    $M \leftarrow \text{zeros}(k, m)$
    for $x \in \mathcal{S}$
        $M \leftarrow \text{Up}_K(M, \mathsf{up}_x)$
    return $M$
```

```
$\text{Up}_K(M, \mathsf{up}_x)$
    $(p_1, \ldots, p_k) \leftarrow R(K, x)$
    for $i \in [k]$
        $M[i][p_i] \mathrel{+}= 1$
    return $M$
```

```
$\text{Qry}_K(M, \mathsf{qry}_x)$
    $(p_1, \ldots, p_k) \leftarrow R(K, x)$
    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 $R : \mathcal{K} \times \mathcal{U} \to [m]^k$ that maps data-object elements (encoded as strings) to a vector of positions in the array $\boldsymbol{M}$. A concrete scheme is given by a particular choice of parameters.

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 [12]. 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). This is equivalent to saying that the stream $\vec{S}$ and the target x are determined before the random choices of the structure are made.

4 HeavyKeeper

```
$\text{Rep}_K(\mathcal{S})$
    // initialise $k \times m$ (fp,cnt) 2-d array
    for $i \in [k]$
        $A[i] \leftarrow [(\star, 0)] \times m$
    for $x \in \mathcal{S}$
        $A \leftarrow \text{Up}_K(A, \mathsf{up}_x)$
    return $A$
```

```
$\text{Qry}_K(A, \mathsf{qry}_x)$
    $(p_1, \ldots, p_k) \leftarrow R(K, x)$
    $\text{fp}_x \leftarrow T(K, x)$
    $\text{cnt}_x \leftarrow 0$
    for $i \in [k]$
        if $A[i][p_i].\text{fp} = \text{fp}_x$
            $\text{cnt} \leftarrow A[i][p_i].\text{cnt}$
            $\text{cnt}_x \leftarrow \max\{\text{cnt}_x, \text{cnt}\}$
    return $\text{cnt}_x$
```

```
$\text{Up}_K(A, \mathsf{up}_x)$
    $(p_1, \ldots, p_k) \leftarrow R(K, x)$
    $\text{fp}_x \leftarrow T(K, x)$
    for $i \in [k]$
        if $A[i][p_i].\text{fp} \notin \{\text{fp}_x, \star\}$
            $r \twoheadleftarrow [0, 1)$
            if $r \leq d^{A[i][p_i].\text{cnt}}$
                $A[i][p_i].\text{cnt} \mathrel{-}= 1$
        // overtake the counter if 0
        if $A[i][p_i].\text{cnt} = 0$
            $A[i][p_i].\text{fp} \leftarrow \text{fp}_x$
        // increase the count if $\text{fp} = \text{fp}_x$
        if $A[i][p_i].\text{fp} = \text{fp}_x$
            $A[i][p_i].\text{cnt} \mathrel{+}= 1$
    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 $R : \mathcal{K} \times \mathcal{U} \to [m]^k$, a function $T : \mathcal{K} \times \mathcal{U} \to \{0, 1\}^n$ for some desired fingerprint length n, decay probability $0 < d \leq 1$, and integers $m, k \geq 0$.

Like CMS, an instance of the HeavyKeeper data structure is parameterized by positive integers k, m, and a function $R\colon \mathcal{K} \times \mathcal{U} \to [m]^k$; in addition, it is parameterized by real-valued $d \in (0,1]$, and *fingerprinting function* $T\colon \mathcal{K} \times \mathcal{U} \to \{0,1\}^n$ 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 $A[i][j].\mathrm{fp} = T(K,z)$ 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 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 $A[i][j].\mathrm{cnt}$ 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 $A[i][j].\mathrm{cnt}$ 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 $\mathrm{fp}_y = T(K,y)$, 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. Writing $(p_1,\ldots,p_k) \leftarrow R(K,x)$ and $\mathrm{fp}_x \leftarrow T(K,x)$, a point-query for x returns $\max\{A[i][p_i].\mathrm{cnt} \mid A[i][p_i].\mathrm{fp} = \mathrm{fp}_x, i \in [k]\}$, i.e., the largest counter value among those positions in A that "remember" having seen x. If that set is empty, the point-query returns 0.

Yang et al. [34] 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 Appendix A, we recover a meaningful result (under their assumptions).

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 $R : \mathcal{U} \to [m]^k$ and $T : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$ via calls to the **Hash**-oracle. In detail, given some unambiguous encoding function $\langle \cdot,\cdot,\cdot \rangle$, for CMS we set $R(K,x) = (\mathbf{Hash}(\langle 1,K,x \rangle), \mathbf{Hash}(\langle 2,K,x \rangle, \ldots, \mathbf{Hash}(\langle k,K,x \rangle)))$, and for HK, we set $R(K,x)[i] = \mathbf{Hash}(\langle \text{"cnt"}, i, K, x \rangle)$ and $T(K,x) = \mathbf{Hash}(\langle \text{"fp"}, k+1, K, x \rangle)$. 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., $\{0,1\}^n$ for some constant $n \geq 0$, when called with "fp".*

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 $\mathcal{C} = \bigcup_{i \in [k]} V_x^i$ contains a set of stream elements that "cover" the counters $M[i][p_i]$ associated to x. Since the presence of a covering $\mathcal{C}$ 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 $\mathcal{C} = \{y_1, y_2, \ldots, y_t\}$ is an $(\mathcal{Y}, x, r)$-*cover *if: (1) $\mathcal{C} \subseteq \mathcal{Y} \setminus \{x\}$, 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 par-

*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.

ticular, HK has a fingerprint function $T(K,\cdot)$, and we define the set $\mathcal{FP}(K,x)=\{y\in\mathcal{U}\mid T(K,y)\neq T(K,x)\}$. We will typically write fp_x as shorthand for the result of computing $T(K,x)$, dropping explicit reference to the key K;

In analyzing their HK structure, Yang et al. [34], 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

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

Note that relation (1) fully defines z for any fixed values of $\Pr[\,L^r\le z\,]$, m,k,r. Thus, we will be able to relate $\Pr[\,\mathsf{Cover}_x^r\,]$ and $\Pr[\,L^r=z\,]$ via the resources used in attacks, e.g., Cover_x^r occurs iff $L^r\le f_{m,k,r}(q_H,q_U,q_Q)$ for some function $f_{m,k,r}$ of the adversarial resources.

When $r=1$, this simplifies to The L_i^1 are geometric random variables with success probability p, and

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

with $q=1-p$. When $r=2$ we arrive at a more complicated expression

$$\Pr[L^2\le z]=(1-zq^{z-1}+(z-1)q^z)^k. \tag{3}$$

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. [†] This constant depends only on parameters m and k.

[†]Concretely, when $k=5,m=1000$ we have $\mathbb{E}[L^1]\approx 2283$. Experimentally, we verified this result over $10,000$ trials with an average of 2281 insertions needed to find a cover set for a per-trial randomly chosen element x.

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.

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 $\mathsf{up}_{e_i}, \ldots, \mathsf{up}_{e_t}$ and $e_1, \ldots, e_t$ is an r-cover for the target. For concision, define random variable $\mathsf{Err} = \mathbf{Atk}_{\Pi}^{\text{err-fe[u,v]}}(\mathcal{A})$. We will mainly focus on $\mathbb{E}[\mathsf{Err}]$ when analyzing the behavior of structures, so here we observe that the non-negative nature of Err allows us to write $\mathbb{E}[\mathsf{Err}] = \sum_{\xi>1} \Pr[\,\mathsf{Err} \geq \xi\,]$. 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 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 $\mathcal{C}$ is found by the first stage (so Cover_x^1 holds), the second stage can insert $\mathcal{C}$ until the resource budget is exhausted. Note that each insertion of $\mathcal{C}$ will increase the CMS estimation-error by one. Our attacks ensure that $|\mathcal{C}| \leq k$, and so the number of $\mathcal{C}$-insertions in the second stage is at least $\left\lfloor \frac{q_U - Q}{k} \right\rfloor$. This implies that $\mathbb{E}[\mathsf{Err} \mid Q] \geq \sum_{\xi=1}^{\lfloor (q_U-Q)/k \rfloor} \Pr\left[\,\mathsf{Err} \geq \xi \mid Q, \mathsf{Cover}_x^1\,\right] \Pr\left[\,\mathsf{Cover}_x^1 \mid Q\,\right]$ 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

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

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; $\mathbb{E}[\mathsf{Err}] = \left\lfloor \frac{q_U}{k} \right\rfloor \Pr\left[\,\mathsf{Cover}_x^1 \mid Q{=}0\,\right]$. 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 (2) we arrive at

$$\Pr\left[\mathsf{Cover}_x^1 \mid Q{=}0\right] = \left(1 - (1{-}1/m)^{\frac{q_H}{k} - 1}\right)^k \tag{4}$$

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 full attack can be found in Figure 4.

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 5) 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

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

```
FindCover^{Hash}(r, x, K)
1:  cover ← $\emptyset$; found ← False
2:  $\mathcal{I} \leftarrow \emptyset$; tracker ← zeros($k$)
3:  // $R(K,x)[i] = \mathbf{Hash}(\langle i, K, x\rangle)$
4:  $(p_1, p_2, \ldots, p_k) \leftarrow R(K, x)$
5:  while not found
6:      if $q_H$ Hash-queries made
7:          return $\emptyset$
8:      $y \twoheadleftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})$
9:      $\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}$
10:     $(q_1, q_2, \ldots, q_k) \leftarrow R(K, y)$
11:     for $i \in [k]$
12:         if $p_i = q_i$ and tracker[$i$] $< r$
13:             cover ← cover $\cup \{y\}$
14:             tracker[$i$] += 1
15:     if sum(tracker) $= rk$
16:         found ← True
17: return cover
```

Figure 4: Cover Set Attack for the CMS in public hash function setting. We use $R(K,x)$ to mean $(\mathbf{Hash}(\langle 1,K,x\rangle), \mathbf{Hash}(\langle 2,K,x\rangle, \ldots, \mathbf{Hash}(\langle k,K,x\rangle)))$. The attack is parametrized with the update and **Hash** query budget q_U and q_H.

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. Say the last added element was z_1. As this caused the CMS estimate to increase, z_1 shares a counter of x. Moreover, any counter covered by z_1 must have been minimal, i.e., still holding its initial value c_i, at the time that z_1 was inserted. Thus, we set our round-one candidate cover $\mathcal{C}_1 \leftarrow \{z_1\}$. 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 $\mathcal{C}_1$ and allowing us to detect an element covering a "fresh" x-counter with the next frequency estimate change.

We then proceed as follows. In each round $i = 2, \ldots$ we first keep reinserting $\vec{I} \setminus \mathcal{C}_{i-1}$ *in order* until some z_i increases the CMS estimate for x again, then we set $\mathcal{C}_i \leftarrow \mathcal{C}_{i-1} \cup \{z_i\}$. We then reinsert the cover until x's estimate stops changing signalling, again, that the minimal x-counters are not covered by $\mathcal{C}_i$ 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 $\mathcal{C}_{i-1}$, and there are only k counters to cover. Let d_i be the number of re-insertions of $\mathcal{C}_i$ in round i. Then, the number of **Up**-queries required to reach $\mathcal{C}_\ell$ is $q'_U \leq \ell|\vec{I}| + \sum_{i=1}^{\ell-1} i d_i$. So, $\mathcal{C}_\ell$ 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 $\mathcal{C}_i$ 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 $|\vec{I}| = L_1$), and simplifying, we get $\mathsf{Err} \geq \left\lfloor \left(\frac{\ell+1}{2} + \frac{1}{\ell}\left(q_U + \sum_{i=1}^{\ell-1} (\ell - i)\delta_i \right) - L^1 \right) \right\rfloor$. We note that Err is a function of several random variables: L^1, ℓ, $\{\delta_i\}_{i\in[\ell-1]}$. 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 $z \in \mathcal{C}$ 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

```
$\text{CoverAttack}^{\mathbf{Up},\mathbf{Qry}}(x,\bot,\bot)$
1 : cover $\leftarrow \text{FindCover}^{\mathbf{Up},\mathbf{Qry}}(x)$
2 : until $q_U$ Up-queries made:
3 :     for $e \in$ cover: $\mathbf{Up}(e)$
4 : return done

$\text{MinUncover}^{\mathbf{Up},\mathbf{Qry}}(x, a', \text{cover})$
1 : $b' \leftarrow -1$
2 : while $a' \neq b'$
3 :     if $(q_U - |\text{cover}| + 1)$Up-
4 :         or $q_Q$ Qry-queries made:
5 :         return cover
6 :     $b' \leftarrow a'$
7 :     for $y \in$ cover : $\mathbf{Up}(y)$
8 :     $a' \leftarrow \mathbf{Qry}(x)$
9 : return $a'$

$\text{FindCover}^{\mathbf{Up},\mathbf{Qry}}(x)$
1 :  // find 1-cover for x
2 :  cover $\leftarrow \emptyset$
3 :  found $\leftarrow$ False
4 :  $\vec{I} \leftarrow \emptyset; a \leftarrow \mathbf{Qry}(x)$
5 : while not found
6 :     if $q_U$ Up- or $q_Q$ Qry-queries made
7 :         return cover
8 :     $y \twoheadleftarrow \mathcal{U} \setminus (\vec{I} \cup \{x\})$
9 :     $\vec{I} \leftarrow \vec{I} \cup \{y\}$
10 :    $\mathbf{Up}(y);\ a' \leftarrow \mathbf{Qry}(x)$
11 :    if $a' \neq a$ :
12 :        cover $\leftarrow \{y\}$
13 :        found $\leftarrow$ True
14 : for $i \in [2, 3, \ldots, k]$
15 :    $a \leftarrow \text{MinUncover}^{\mathbf{Up},\mathbf{Qry}}(x, a', \text{cover})$
16 :    if $a =$ cover : return cover
17 :    for $y \in \mathcal{I}$ // in order of insertion to $\mathcal{I}$
18 :        if $q_U$ Up- or $q_Q$ Qry-queries made
19 :            return cover
20 :        $\mathbf{Up}(y); a' \leftarrow \mathbf{Qry}(x)$
21 :        if $a' \neq a$ :
22 :            cover $\leftarrow$ cover $\cup \{y\}$
23 :            $\vec{I} \leftarrow \mathcal{I} \setminus \{y\}$
24 :            break
25 : return cover
```

Figure 5: 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.

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 Appendix B.

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 (Figure 6). 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 $\mathcal{C}$ is found. By the definition of L^1, the cover is found with ($q'_U = 1 + L^1$) **Up**-queries, and is after reinserted $\lfloor (q_U - q'_U)/|\mathcal{C}| \rfloor$ times, each time adding one to the estimation error. Hence, $\mathsf{Err} \geq \lfloor (q_U - 1 - L^1)/|\mathcal{C}| \rfloor \geq \lfloor (q_U - 1 - L^1)/k \rfloor$ and $\mathbb{E}[\mathsf{Err}] \geq \frac{q_U - 1 - \mathbb{E}[L^1]}{k} \approx \frac{q_U - mH_k}{k}$.

```
$\text{CoverAttack}^{\mathbf{Up}}(x, \bot, \mathsf{repr})$
cover $\leftarrow \text{FindCover}^{\mathbf{Up}}(1, x, \mathsf{repr})$
**until** $q_U$ **Up**-queries made:
    **for** $e \in$ cover: $\mathbf{Up}(e)$
**return** done
```

```
$\text{FindCover}^{\mathbf{Up}}(r, x, \mathsf{repr})$
cover $\leftarrow \emptyset$; found $\leftarrow \mathsf{False}$
$\mathcal{I} \leftarrow \emptyset$; tracker $\leftarrow$ zeros$(k)$
$\mathsf{repr}' \leftarrow \mathbf{Up}(x)$
// compute $x$'s indices
**for** $i \in [k]$
    **for** $j \in [m]$
        **if** $\mathsf{repr}'[i][j] \neq \mathsf{repr}[i][j]$
            $p_i \leftarrow j$; **break**;
**while** not found
    **if** $q_U$ **Up**-queries made : **return** $\emptyset$
    $y \twoheadleftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})$
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}$
    $\mathsf{repr} \leftarrow \mathsf{repr}'$
    $\mathsf{repr}' \leftarrow \mathbf{Up}(y)$
    // compute $y$'s indices
    **for** $i \in [k]$
        **for** $j \in [m]$
            **if** $\mathsf{repr}'[i][j] \neq \mathsf{repr}[i][j]$
                $q_i \leftarrow j$; **break**;
    **for** $i \in [k]$
        // compare $x$'s and $y$'s indices row by row
        **if** $p_i = q_i$ **and** tracker$[i] < r$
            cover $\leftarrow$ cover $\cup \{y\}$
            tracker$[i] \mathrel{+}= 1$
    **if** $\mathsf{sum}$(tracker) $= rk$
        found $\leftarrow \mathsf{True}$
**return** cover
```

Figure 6: Cover Set Attack for the CMS in private hash function and public representation setting. The attack is parametrized with the update query budget q_U.

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, overestimations 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 an $(\mathcal{FP}_x, x, r)$-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. (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 $n_x \gg 0$. (At least one such counter would need to be decremented to zero, in order for ownership of that counter to be transferred to x, i.e., $A[i][p_i].\text{fp} \leftarrow \text{fp}_\text{x}$.)

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 [22]. 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 $\mathcal{C} = \{z_1, z_2 \dots\}$ and then insert it t times, with the value for t set to ensure the target counters will almost certainly *not* be decremented in future. We set a value for t in a way the probability of decrementing any of the target's counters at some point set to t, with fingerprint $\neq \mathrm{fp}_x$, is $< p$, with sufficiently small $0 < p < 1$. We use $p = 2^{-128}$ in our experiments.

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 $(D^t) = \bigvee_{i=1} D_i^t$. Then, $\Pr[D_i^t] \leq \binom{q_U}{t} \prod_{j=1}^{t} d^j \leq (q_U)^t \, d^{\frac{t(t+1)}{2}}$. Let $f(t) = k\,(q_U)^t \, d^{\frac{t(t+1)}{2}}$. 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 $f(t) < p$.

Public hash and public representation setting. This attack (Figure 7) is the CMS attack for the public hash setting, but with 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 (4)) obtaining

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

Private hash and private representation setting. We present the attack for this setting in Figure 8. The attack 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 the last). So, we set our round-one candidate cover set $\mathcal{C}_1 \leftarrow \{z_1\}$ and insert t times to the HK. Now we are at the point when all $c_1, \dots, c_k$ are "owned" by elements $\neq x$, and, under the NFC condition, 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 $i = 2, \dots$ 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 $c_1, \dots, c_k$ 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 $\mathcal{C}_i$, by inserting *new distinct* elements, until we find a z_i that drops HK(x) to 0. We set $\mathcal{C}_i \leftarrow \mathcal{C}_{i-1} \cup \{z_i\}$ and insert z_i t times. At this point, all counters $c_1, \dots, c_k$ are "owned" by elements $\neq x$ again, and all are of value one, but the ones covered by $\mathcal{C}_i$ which (very likely) hold a value strictly greater than 1 and (very) close to t.

```
$\text{CoverAttack}^{\mathbf{Hash},\mathbf{Up},\mathbf{Qry}}(x, K, \mathsf{repr})$
cover $\leftarrow$ FindCover$^{\mathbf{Hash}}(x, K)$
$t \leftarrow$ Get-t(|cover|)
for $e \in$ cover
    for $i \in [t]$: $\mathbf{Up}(e)$
until $q_U$ Up-queries made:
    $\mathbf{Up}(x)$
return done

Get-t()
$g(t) \leftarrow \log_2(k \cdot (q_U)^t\, d^{t(t+1)/2}) - \log_2(p)$
// find the roots of the negative quadratic polynomial g
$t_1, t_2 \leftarrow$ FindRootsOf($g$) // $t_1 \leq t_2$
// set $t$ so $t \geq 1$ and $g(t) < 0$
if $t_1 > 1$ or $t_2 < 1 : t \leftarrow 1$
if $t_2 > 1 : t \leftarrow \lceil t_2 \rceil$
if $t_2 = 1 : t \leftarrow 2$
return $t$
```

```
$\text{FindCover}^{\mathbf{Hash}}(x, K)$
cover $\leftarrow \{\}$; found $\leftarrow$ False
$\mathcal{I} \leftarrow \emptyset$; tracker $\leftarrow$ zeros($k$)
// $R(K, x)[i]$
// $= \mathbf{Hash}(\langle\text{“}ct\text{”}, i, K, x\rangle)$
$(p_1, p_2, \ldots, p_k) \leftarrow R(K, x)$
while not found
    if $q_H$ Hash-queries made
        return $\emptyset$
    $y \twoheadleftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})$
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}$
    $(q_1, q_2, \ldots, q_k) \leftarrow R(K, y)$
    for $i \in [k]$
        if $p_i = q_i$
            // remove duplicates
            cover[$i$] $\leftarrow y$
            if tracker[$i$] $< 1$
                tracker[$i$] $\leftarrow 1$
    if sum(tracker) $= k$
        found $\leftarrow$ True
// return the cover
return cover.values()
```

Figure 7: Cover Set Attack for the HK in public hash function setting. We use $R(K, x)$ to mean $(\mathbf{Hash}(\langle\text{“}ct\text{”}, 1, K, x\rangle), \mathbf{Hash}(\langle\text{“}ct\text{”}, 2, K, x\rangle, \ldots, \mathbf{Hash}(\langle\text{“}ct\text{”}, k, K, x\rangle)))$. The attack is parametrized with the update and **Hash** query budget q_U and q_H.

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 element new element to incomplete cover $\mathcal{C}_{i-1}$. 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 $\mathcal{C}_{i-1}$. This is because all the counters covered by $\mathcal{C}_{i-1}$ were set to value t (or a value close to t with very high probability[‡]) 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 1-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

$$q_U' \leq \sum_{i=1}^{k}(d_i + |\mathcal{I}_i| + t) = \sum_{i=1}^{k}(d_i + |\mathcal{I}_i|) + tk. \tag{5}$$

So, x can be potentially inserted $q_U - q_U'$ times, accumulating some additional error[§]. Let us assume that q_Q is not the limiting resource in the attack. Say $\mathcal{C}$ is the attack's maximal round candidate cover. Whenever $\neg(D^t)$, adding z_i t times to the HK incremented one of the x's counters, not yet set to value t by elements in $\mathcal{C}_{i-1}$. If, in addition, we have $|\mathcal{C}| = k$, 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 $\mathcal{C}$ are

[‡]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 the probability ≥ 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.

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

```
CoverAttack^{Up,Qry}(x, ⊥, ⊥)
1 : FindInsertCover^{Up,Qry}(x)
2 : until $q_U$ Up-queries made:
3 :    Up($x$)
4 : return done

Reintro^{Up,Qry}(x)
1 : // reintroduce target $x$
2 : while True
3 :    if $q_U$ Up- or $q_Q$ Qry-queries made:
4 :          return
5 :    Up($x$); $a \leftarrow$ Qry($x$)
6 :    if $a > 0$ : return
7 : endwhile
8 :

FindInsertCover^{Up,Qry}(x)
1 : // insert $\leq k$ elements $t$ times in a row
2 : cover $\leftarrow \emptyset$
3 : $t \leftarrow$ Get-t()
4 : $\mathcal{I} \leftarrow \emptyset$
5 : for $i \in [1, 2, \ldots, k]$
6 :    Reintro^{Up,Qry}($x$)
7 :    while True
8 :       if $q_U$ Up- or $q_Q$ Qry-queries made
9 :          return
10 :      $y \twoheadleftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})$
11 :      $\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}$
12 :      Up($y$); $a \leftarrow$ Qry($x$)
13 :      if $a = 0$ :
14 :         cover $\leftarrow$ cover $\cup \{y\}$
15 :         for $j \in [t]$ : Up($y$)
16 :         break
17 : return
```

Figure 8: Cover Set Attack for the HK in private hash function and representation setting. The attack is parametrised with the update and query query budget q_U and q_Q. The attack uses the function Get-t(.) from Figure 7.

completed every further insertion of x ($q_U - q'_U$ of them) increased the error by 1. Note that $|\mathcal{C}| = k$ implies the attack completed exactly k rounds and

$$\begin{aligned}&\left[\mathsf{Err} \mid \neg\left(D^t\right), |\mathcal{C}| = k\right] \\ &\geq \sum_{i=1}^{k}(d_i) + q_U - \sum_{i=1}^{k}\left(d_i + \left[|\mathcal{I}_i| \mid \neg\left(D^t\right), |\mathcal{C}| = k\right]\right) - tk \\ &\geq q_U - \sum_{i=1}^{k}\left[|\mathcal{I}_i| \mid \neg\left(D^t\right), |\mathcal{C}| = k\right] - tk.\end{aligned}$$

Let D_i be the set of rows j with $A[j][p_j].\mathrm{fp} = \mathrm{fp}_x$ (i.e. x "owning" the counter), and let $c_{i,j}$ be 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 from x in row $j \in D_i$ 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. Moreover, $c_{i,j} = 1$ for all $j \in D_i$ – counters "owned" by x equal 1 after every reinsertion step. As $|D_1| = k$ we have that $|\mathcal{I}_1| = \max_{j \in D_1}\{Y_{1,j}\}$ 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. geometric variables with

$p = \frac{d}{m}$, we have that $\mathbb{E}\left[|\mathcal{I}_i|\right] \le \mathbb{E}\left[|\mathcal{I}_1|\right]$. So, $\mathbb{E}\left[|\mathcal{I}_i|\right] \le \frac{m}{d} H_k$. This implies that

$$\begin{aligned}
\mathbb{E}\left[\mathsf{Err}\right] &= \sum_{s=0}^{k} \mathbb{E}\left[\mathsf{Err} \mid \left(D^t\right), |\mathcal{C}| = s\right] \Pr\left[\left(D^t\right) \wedge |\mathcal{C}| = s\right] \\
&\quad + \sum_{s=0}^{k} \mathbb{E}\left[\mathsf{Err} \mid \neg\left(D^t\right), |\mathcal{C}| = s\right] \Pr\left[\neg\left(D^t\right) \wedge |\mathcal{C}| = s\right] \\
&\ge \mathbb{E}\left[\mathsf{Err} \mid \neg\left(D^t\right), |\mathcal{C}| = k\right] \Pr\left[\neg\left(D^t\right) \wedge |\mathcal{C}| = k\right] \\
&\ge (q_U - tk) \Pr\left[\neg\left(D^t\right) \wedge |\mathcal{C}| = k\right] - \sum_{i=1}^{k} \mathbb{E}\left[|\mathcal{I}_i|\right] \\
&\ge (q_U - tk) \Pr\left[\neg\left(D^t\right) \wedge |\mathcal{C}| = k\right] - \frac{km}{d} H_k.
\end{aligned}$$

We expect $\Pr\left[\neg\left(D^t\right) \wedge |\mathcal{C}| = k\right] \approx 1$ 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.

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 setting all x's buckets fingerprints. Then, we keep adding *distinct* elements, until all the $A[i][p_i]$.fp has changed, signaling the cover for x has been found. We give a pseudocode description of this attack in Figure 9.

Under the NFC condition, inserting distinct elements keeps all x-counters at value 1. Hence, adding any $y \neq x$ has $\frac{d}{m}$ probability to change $A[i][p_i]$.fp. Let Y_i be the minimal number of distinct element $\neq x$ insertions before $A[i][p_i]$.fp changes from fp_x. 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. Say q_U is the total **Up**-query budget. After the cover finding step, we insert cover $\mathcal{C}$ t times, followed by $q_U - q'_U - t|\mathcal{C}|$ insertions of x. Each x-insertion added one to the error if $\neg\left(D^t\right)$ and

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

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 10 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.

```
CoverAttack$^{\mathbf{Up},\mathbf{Qry}}(x, \perp, \mathsf{repr})$
    cover ← FindCover$^{\mathbf{Up}}(x, \mathsf{repr})$
    $t \leftarrow$ Get-t()
    for $e \in$ cover
        for $i \in [t]$: $\mathbf{Up}(e)$
    until $q_U$ Up-queries made:
        $\mathbf{Up}(x)$
    return done
```

```
FindCover$^{\mathbf{Up}}(x, \mathsf{repr})$
    cover ← {}; found ← False
    $\mathcal{I} \leftarrow \emptyset$; tracker ← zeros($k$)
    $\mathsf{repr}' \leftarrow \mathbf{Up}(x)$
    // compute $x$'s indices
    for $i \in [k]$
        for $j \in [m]$
            if $\mathsf{repr}'[i][j].\mathrm{fp} \neq \mathsf{repr}[i][j].\mathrm{fp}$
                $p_i \leftarrow j$; break;
while not found
        if $q_U$ Up-queries made : return $\emptyset$
        $y \twoheadleftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})$
        $\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}$
        $\mathsf{repr} \leftarrow \mathsf{repr}'$
        $\mathsf{repr}' \leftarrow \mathbf{Up}(y)$
        // compute $y$'s indices
        for $i \in [k]$
            $q_i \leftarrow$ False
            for $j \in [m]$
                if $\mathsf{repr}'[i][j].\mathrm{fp} \neq \mathsf{repr}[i][j].\mathrm{fp}$
                    $q_i \leftarrow j$; break;
        for $i \in [k]$
            // compare $x$'s and $y$'s indices row by row
            if $q_i \neq$ False  and  $p_i = q_i$
                // remove duplicates
                cover[$i$] ← $y$
                if tracker[$i$] < 1
                    tracker[$i$] ← 1
        if sum(tracker) = $k$
            found ← True
    // cover elements 'own' the target's counters
    return cover.values() // all counters set to 1
```

Figure 9: Cover Set Attack for the HK in private hash function and public representation setting. The attack is parametrized with the update query budget q_U. The attack uses the function Get-t(.) from Figure 7.

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 $M \leftarrow \mathrm{UP}_K^{\mathrm{CMS}}(M, \mathsf{up}_x)$ and $A \leftarrow \mathrm{UP}_K^{\mathrm{HK}}(M, \mathsf{up}_x)$, 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 $p_i = R(K, z)[i]$.

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

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

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

Figure 10: Keyed structure $\text{CK}[R, T, m, k]$ supporting point-queries for any potential stream element x (qry_x). $\textsc{Qry}_K^{\text{CMS}}, \textsc{Up}_K^{\text{CMS}}$, resp. $\textsc{Qry}_K^{\text{HK}}, \textsc{Up}_K^{\text{HK}}$, 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 $m, k \geq 0$. A concrete scheme is given by a particular choice of parameters.

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 NFC 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).

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 the value of 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 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 $A[i][p_i].\text{cnt} = 1$ and some $y \neq x$ was the last insertion. Thus, $A[i][p_i] - 1$ is a lowerbound 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, $n_x + A[i][p_i] - 1 \le \sum_{y\in V_x^i} n_y$, which implies that $M[i][p_i] = n_x + \sum_{y\in V_x^i} n_y \le 2n_x + A[i][p_i] - 1$. 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 Appendix C.

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 $n_x \le \frac{M[i][p_i] - A[i][p_i].\text{cnt} + 1}{2} = \Theta_1^i$.* ⧫

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

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 $A[i][p_i]$.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 $\sum_{y\in V_x^i} n_y \ge n_x - A[i][p_i]$.cnt occurrences of $y \in V_x^i$. This implies $M[i][p_i] \ge 2n_x - A[i][p_i]$.cnt, and, by rearranging, $n_x \le \frac{M[i][p_i] + A[i][p_i].\text{cnt}}{2}$. This sketches a proof of the following lemma, whose full proof appears in Appendix C.

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

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

Precise estimation when some $|V_x^i| \in \{0,1\}$. If there exists an i such that $\left|V_x^i\right| = 0$, then $M[i][p_i] = A[i][p_i] = n_x$. Hence, in this special case, both $\mathrm{CMS}(x) = n_x$ and $\mathrm{HK}(x) = n_x$. When this is not the case, $n_x < M[i][p_i]$ for all $i \in [k]$, so $n_x < \mathrm{CMS}(x)$. For CK, on the other hand, if there exists a row i such that $|V_x^i| = 1$, we still have $\mathrm{CK}(x) = n_x$. 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 $\mathrm{CK}(x) = \lfloor \min\{\Theta_1, \Theta_2\} \rfloor = n_x$. The proof can be found in Appendix C.

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

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

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

Finally, we note one more case when $\mathrm{CK}(x) = n_x$. If one of the x's buckets holds uninitialized fingerprint, i.e. $i \in [k]$ such that $A[i][p_i].\mathrm{fp} = \star$, 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 $\mathrm{CMS}(x) = M[i^*][p_{i^*}]$. Then either $\mathrm{CK}(x) = n_x$ or $\left(i^* \in \hat{I}_x\right) \Rightarrow \left(\Theta_1 \le \frac{\mathrm{CMS}(x)}{2}\right)$.* $\blacklozenge$

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

Next, a similar theorem relating 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 $\mathrm{CMS}(x) = M[i^*][p_{i^*}]$. If $i^* \in I_x$ then $\Theta_2 \le \frac{\mathrm{CMS}(x)+\mathrm{HK}(x)}{2}$.* $\blacklozenge$

Proof. We have, $\Theta_2 = \min_{i \in I_x} \Theta_2^i \le \Theta_2^{i^*} = \frac{M[i^*][p_i^*] + A[i^*][p_i^*].\mathrm{cnt}}{2}$. Thus, the theorem is proved by observing that $A[i^*][p_i^*].\mathrm{cnt} \le \mathrm{HK}(x)$. $\square$

Now, if $\mathrm{CK}(x)$ is determined by line 10 of Figure 10, then $\mathrm{CK}(x) = \frac{\mathrm{CMS}(x)+\mathrm{HK}(x)}{2}$. On the other hand, if $\mathrm{CK}(x)$ is determined by line 15, then $\mathrm{CK}(x) = 0 \le \frac{\mathrm{CMS}(x)+\mathrm{HK}(x)}{2}$. If neither of these holds, $\mathrm{CK}(x) = \lfloor \min\{\Theta_1, \Theta_2\} \rfloor$. Thus, Theorem 1 and 2 imply $\lfloor \min\{\Theta_1, \Theta_2\} \rfloor \le \frac{\mathrm{CMS}(x)+\mathrm{HK}(x)}{2}$, giving us the following lemma.

Lemma 3. *For any $x \in \mathcal{U}$, $\mathrm{CK}(x) \le \frac{\mathrm{CMS}(x)+\mathrm{HK}(x)}{2}$.* $\blacklozenge$

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 $\mathrm{CK}(x) - n_x \le \frac{\mathrm{CMS}(x)-\mathrm{HK}(x)}{2}$.* $\blacklozenge$

Consequences of Corollary 2. First, as CMS(x) and 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 CK(x) approaches *zero*.

Next, because CMS is a count-all structure, the *worst case* guarantee is that the error $\mathrm{CK}(x) - n_x \leq \mathrm{CMS}(x)/2$, i.e., when $\mathrm{HK}(x) = 0$. 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 $A[i][p_i]$.cnt to which x maps. As $M[i][p_i]$ observes the same substream as $A[i][p_i]$, and $\mathrm{CMS}(x) = \min_{i\in[k]}\{M[i][p_i]\}$, 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 CMS(x) be very large when 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 [6,9,12,24,25,28,34]. 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¶. We also sourced an additional stream by processing a large English language novel from Project Gutenberg.

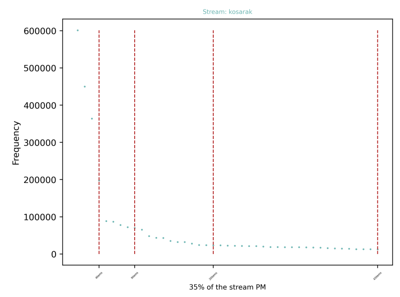
We summarize each of these three streams and why they are of particular interest to experiment on below.

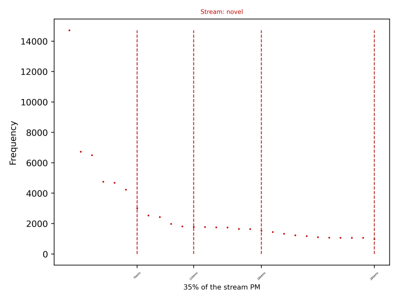
1. **Kosarak Stream:** This data collection contained anonymized click-rate data collected from visits to an online Hungarian news site. The resultant stream is of total length $8,019,015$ with $41,270$ distinct elements. As aforementioned, we sourced this stream from a frequent item mining dataset repository 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.

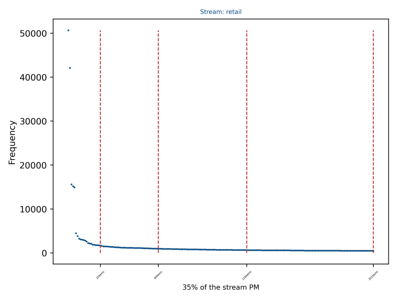
2. **Novel Stream:** We created a stream by processing the individual words sequentially of The Project Gutenberg eBook plaintext edition of the 1851 English-language novel *Moby-Dick; or, The Whale* by Herman Melville (ignoring capitalization and non-alphabetical characters) [27]. Long bodies of natural language obey an approximate Zipf distribution as the frequency of any word is inversely proportional to its rank in an ordered frequency list [4]. It is of interest to measure compact frequency estimators performance against data following a Zipf distribution [6, 9, 13, 25, 28, 34]. 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.

¶`http://fimi.uantwerpen.be/data/`







Kosarak stream | Novel stream | Retail stream

Figure 11: 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 stream, 22 for the Novel stream, and 22 for the Retail stream.

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.

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 [32] 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 [24, 28, 34].

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 [34] 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 [20]. 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 [9].

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 11. 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 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 [31]. 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 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 [34]. It is defined as 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[‖]. 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.

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 [3] and satisfies the desired properties of the exponential decay function stated in [34].

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.

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

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

Table 1: A summary of non-adversarial setting results between the CK, CMS, and HK compact frequency estimators.

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 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 11. 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.

A 2-cover $\mathcal{C}$ for x contains elements $\{y_1, y_2, \ldots, y_t\}$ such that for every counter x maps to in positions $(p_1, p_2, \ldots, p_k) \leftarrow R(K, x)$ it is such that two distinct elements in $\mathcal{C}$ cover each counter. In our attack model we assume an initially empty representation and we never insert x in any of our attacks (except for once to discover its counter positions in the public representation, private hash setting).

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**), 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 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.

Public hash and representation setting. As our other attacks (for CMS and HK) in this setting, the CK attack (Figure 12) can be viewed as a two-step process. 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

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

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

```
FindCover^{Hash}(r, x, K)
  cover ← $\emptyset$; found ← False
  $\mathcal{I} \leftarrow \emptyset$; tracker ← zeros(k)
  // $R(K, x)[i] = \mathbf{Hash}(\langle i, K, x\rangle)$
  $(p_1, p_2, \ldots, p_k) \leftarrow R(K, x)$
  while not found
      if $q_H$ Hash-queries made
          return $\emptyset$
      $y \twoheadleftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})$
      $\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}$
      $(q_1, q_2, \ldots, q_k) \leftarrow R(K, y)$
      for $i \in [k]$
          if $p_i = q_i$ and tracker[i] < r
              cover ← cover $\cup \{y\}$
              tracker[i] += 1
      if sum(tracker) = rk
          found ← True
  return cover
```

Figure 12: Cover Set Attack for the CK in public hash function setting. The attack is parametrized with the update and **Hash** query budget q_U and q_H.

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 $\mathcal{C}$ 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 attack for the setting is presented in Figure 13. The attack begins by querying x to learn its current frequency estimate (= 0 as the filter is initially empty). Say $\left\lfloor \frac{M[1][p_1] - A[1][p_1].\text{cnt}+1}{2} \right\rfloor = c_1, \ldots, \left\lfloor \frac{M[k][p_K] - A[k][p_k].\text{cnt}+1}{2} \right\rfloor = c_k$ be the rows frequency-estimates associated with x. Currently, $\min_{i \in [k]}\{c_i\} = a \geq 0$, but none of the rows' frequency-estimate values was actually used to obtain CK(x) on an empty filter, storing no fingerprints.

The attack then keeps inserting distinct random elements, checking until CK(x) increases to $a+1$, implying that (1) all the fingerprints $A[i][p_i]$.fp are set and (2) a two cover for x was inserted as a direct result of Corollary 1. Let $\vec{I}$ be the stream of inserted elements at the moment that this happens. We next begin the first "round" of extracting a $(\vec{I}, x, 2)$ cover from $\vec{I}$. Let z_1 be the last inserted element. Inserting z_1 caused CK(x) to increase, and z_1 must have increased one of the minimal row-frequency estimates from 0 to 1 (by covering the counter for the second time). So, we set our round-one candidate $(\vec{I}, x, 2)$ cover as $\mathcal{C}_1 \leftarrow \{z_1\}$.

Let $\mathcal{M}(\mathcal{C}_1) = \{i \in [k] \mid \exists z_1, z_2 \in \mathcal{C} : R(K, z_1)[i] = R(K, z_2)[i] = p_i\}$, i.e., the set of rows whose x-counters are 2-covered by $\mathcal{C}_1$, and let $d = \min_{j \notin \mathcal{M}(\mathcal{C}_1)}\{c_j\} - \min_{i \in \mathcal{M}(\mathcal{C}_1)}\{c_i\}$. Notice that reinserting $\mathcal{C}_1$ increases all c_j by one per reinsertion, and keeps each c_i at the same value (if none of elements in $\mathcal{C}_1$ cover the counter, or if exactly one element from $\mathcal{C}_1$ covers the counter, and, in addition, also owns the counter) or, when the exactly one element from $\mathcal{C}_1$ covering the counter does not own it, first only limited (i.e. $A[i][p_i]$.cnt) number of reinsertions increase c_i by one, and the c_i stops changing.

At the point when CK(x) stops changing we have that the rows associated with the currently minimal

```
CoverAttack^{Up,Qry}(x, ⊥, ⊥)
1 : cover ← FindCover^{Up,Qry}(x)
2 : until $q_U$ Up-queries made:
3 :     for $e \in$ cover: Up($e$)
4 : return done

MinUncover^{Up,Qry}(x, a', cover)
1 : $b' \leftarrow a' - 1$
2 : while $a' \neq b'$
3 :     if $(q_U - |\text{cover}| + 1)$Up-
4 :         or $q_Q$ Qry-queries made:
5 :             return cover
6 :     $b' \leftarrow a'$
7 :     for $y \in$ cover : Up($y$)
8 :     $a' \leftarrow$ Qry($x$)
9 : return $a'$

FindCover^{Up,Qry}(x)
1 :  // find 2- cover for x
2 :  cover ← $\emptyset$
3 :  found ← False
4 :  $\mathcal{I} \leftarrow \emptyset; a \leftarrow$ Qry($x$)
5 : while not found
6 :      if $q_U$ Up- or $q_Q$ Qry-queries made
7 :          return cover
8 :      $y \twoheadleftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})$
9 :      $\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}$
10 :     Up($y$);  $a' \leftarrow$ Qry($x$)
11 :     if $a' \neq a$ :
12 :         cover ← $\{y\}$
13 :         found ← True
14 : for $i \in [2, 3, \dots, 2 \cdot k]$
15 :     $a \leftarrow$ MinUncover^{Up,Qry}($x, a'$, cover)
16 :     if $a$ = cover : return cover
17 :     for $y \in \mathcal{I}$ // in order of insertion to $\mathcal{I}$
18 :         if $q_U$ Up- or $q_Q$ Qry-queries made
19 :             return cover
20 :         Up($y$); $a' \leftarrow$ Qry($x$)
21 :         if $a' \neq a$ :
22 :             cover ← cover $\cup \{y\}$
23 :             $\mathcal{I} \leftarrow \mathcal{I} \setminus \{y\}$
24 :             break
25 : return cover // cover is inserted at least once
```

Figure 13: Cover Set Attack for the CK in private hash function and representation setting. The attack is parametrized with the update and query query budget q_U and q_Q.

row frequency-estimates of x have not been 2-covered by $\mathcal{C}_1$. This implies that the next element to increase CK(x) must cover a counter not yet 2-covered by $\mathcal{C}_1$.

At this point we begin round 2, searching for $z_2 \in \vec{I} \setminus \mathcal{C}_1$ that covers a row associated with the newly minimal row frequency-estimate of x. Recall that, by definition, $\vec{I}$ certainly contains a 2-cover. So, we are guaranteed to hit an element satisfying $z_2 \neq z_1$ that increases the estimate after its insertion. Thus, to find z_2 we keep adding $z \in \vec{I} \setminus \mathcal{C}_1$ in order until CK(x) stops changing. At the point this happens, the last inserted element is the element we are looking for and we set z_2 to the element, and our round-2 candidate $(\vec{I}, x, 2)$ cover as $\mathcal{C}_2 \leftarrow \mathcal{C}_1 \cup \{z_2\}$. Continuing this this way, after $\ell \leq 2k$ rounds we will have found a complete 2-cover for x. As each round i adds exactly one new element z_i to the incomplete cover $\mathcal{C}_{i-1}$, and there can only be k counters to 2-cover, maximally $2k$ rounds will be executed before finding a $(\vec{I}, x, 2)$-cover.

Notice that in round ℓ, when we reinsert $\mathcal{C}_\ell$ we will never observe that some new row started to hold the new minimal row frequency-estimate of x: all x-counters are 2-covered by $\mathcal{C}_\ell$, so all will be increased by each reinsertion. Nonetheless, each reinsertion of $\mathcal{C}_\ell$ adds one to the estimation error, and these reinsertions may continue until the resource budget is exhausted, i.e., until a total of q_U elements have been inserted (via **Up**) as part of the attack.

Say each round inserts round-i 2-cover set candidate $\mathcal{C}_i$ $\delta_i + 1$ times. Then, by definition, each complete round increases the error by δ_i by reinserting $\mathcal{C}_i$, and additionally by one when z_i has been detected. So, the number of **Up**-queries (i.e. insertions) required to reach the complete cover $\mathcal{C}_\ell$ is

$$q'_U \leq \ell|\vec{I}| + \sum_{i=1}^{\ell-1} i(\delta_i + 1)$$

and so $\mathcal{C}_\ell$ can potentially be reinserted at least $\lfloor (q_U - q'_U)/\ell \rfloor$ times, each time adding one to the estimation error. Assuming q_Q is not the limiting factor, the error introduced by the attack is

$$\mathsf{Err} \geq \left\lfloor \left(\frac{\ell+1}{2} + \frac{1}{\ell}\left(q_U + \sum_{i=1}^{\ell-1} (\ell - i)\delta_i \right) - L^2 \right) \right\rfloor .$$

The error bound we derive is similar to the one we derived for the our CMS attack, but with L^1 replaced with L^2 as now $|\vec{I}|$ is precisely L^2. As for the CMS attack, for reasonable sizes of the CK we mainly expect $\ell = 2k$ and that $\frac{1}{2k}\sum_{i=1}^{2k-1}(2k-i)\mathbb{E}[\delta_i] = O(2k)$. Given that $k \ll m$ in all practical settings, we expect

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

to approximate $\mathbb{E}[\mathsf{Err}]$.

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

Private hash and public representation setting. This attack (Figure 14) 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

	Public Hash Setting			**Private Hash, Private Rep Setting**		
Structure	|cover|	Experimental Err	$\mathbb{E}[\mathsf{Err}]$	|cover|	Experimental Err	$\mathbb{E}[\mathsf{Err}]$
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 $\mathbb{E}[\mathsf{Err}]$ according to our analysis.

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.

```
CoverAttack^{Up,Qry}$(x, \perp, \mathsf{repr})$
  cover ← FindCover^{Up}$(2, x, \mathsf{repr})$
  until $q_U$ Up-queries made:
    for $e \in$ cover: Up$(e)$
  return done
```

```
FindCover^{Up}$(r, x, \mathsf{repr})$
  $\langle M, A\rangle \leftarrow \mathsf{repr}$
  cover ← $\emptyset$; found ← False
  $\mathcal{I} \leftarrow \emptyset$; tracker ← zeros$(k)$
  $\langle M', A'\rangle \leftarrow$ Up$(x)$
  // compute $x$'s indices
  for $i \in [k]$
    for $j \in [m]$
      if $M'[i][j] \neq M[i][j]$
        $p_i \leftarrow j$; break;
  while not found
    if $q_U$ Up-queries made : return $\emptyset$
    $y \twoheadleftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})$
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}$
    $\langle M, A\rangle \leftarrow \langle M', A'\rangle$
    $\langle M', A'\rangle \leftarrow$ Up$(y)$
    // compute $y$'s indices
    for $i \in [k]$
      for $j \in [m]$
        if $M'[i][j] \neq M[i][j]$
          $q_i \leftarrow j$; break;
    for $i \in [k]$
      // compare $x$'s and $y$'s indices row by row
      if $p_i = q_i$ and tracker$[i] < r$
        cover ← cover $\cup \{y\}$
        tracker$[i] += 1$
    if sum(tracker) $= rk$
      found ← True
  return cover
```

Figure 14: Cover Set Attack for the CK in private hash function and public representation setting. The attack is parametrized with the update query budget q_U.

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 CK(x) is largest when HK$(x) \ll$ CMS(x). In particular, when x does not own any of its counters 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 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 $\Delta = |\text{CK}(x) - n_x|$ be the potential error in the estimate CK(x). Dropping the floor for brevity, if CK$(x) = \Theta_1$ then Lemma 2 tells us that

```
Rep_K(S)
    M ← zeros(k, m)
    for i ∈ [k]
        A[i] ← [(⋆, 0)] × m
    repr ← ⟨M, A⟩
    for x ∈ S
        repr ↞ Up_K(repr, up_x)
    return repr

Up_K(repr, up_x)
    ⟨M, A⟩ ← repr
    M ↞ Up_K^CMS(M, up_x)
    A ↞ Up_K^HK(A, up_x)
    return repr ← ⟨M, A⟩

Qry_K(repr, qry_x)
    ⟨M, A⟩ ← repr
    (p_1, ..., p_k) ← R(K, x), fp_x ← T(K, x)
    Θ_1, Θ_2, Δ ← ∞
    flag ← False
    N ← $\sum_{j=1}^{m} M[1][j]$
    $\mathrm{cnt}_{\mathrm{UB},x}$ ← QryCMS_K(M, qry_x)
    $\mathrm{cnt}_{\mathrm{LB},x}$ ← $\mathrm{Qry}_K^{\mathrm{HK}}(A, \mathsf{qry}_x)$
    if $\mathrm{cnt}_{\mathrm{UB},x} = \mathrm{cnt}_{\mathrm{LB},x}$
        return $\mathrm{cnt}_{\mathrm{UB},x}$, flag
    for i ∈ [k]
        if A[i][p_i].fp = ⋆
            $\mathrm{cnt}_{\mathrm{UB},x}$ ← 0
            return 0, flag
        else if A[i][p_i].fp ≠ fp_x
            Θ ← $\frac{M[i][p_i] - A[i][p_i].\mathrm{cnt} + 1}{2}$
            Θ_1 ← min {Θ_1, Θ}
            $\hat{\Delta}$ ← $\frac{M[i][p_i] - A[i][p_i].\mathrm{cnt} + 1}{2}$
            Δ ← min {Δ, $\hat{\Delta}$}
        else if A[i][p_i].fp = fp_x
            Θ ← $\frac{M[i][p_i] + A[i][p_i].\mathrm{cnt}}{2}$
            Θ_2 ← min {Θ_2, Θ}
            $\hat{\Delta}$ ← $\frac{M[i][p_i] - A[i][p_i].\mathrm{cnt}}{2}$
            Δ ← min {Δ, $\hat{\Delta}$}
    $\mathrm{cnt}_{\mathrm{UB},x}$ ← ⌊min {Θ_1, Θ_2}⌋
    if Δ ≥ ψN
        flag ← True
    return $\mathrm{cnt}_{\mathrm{UB},x}$, flag
```

Figure 15: Keyed structure CK$[R, T, m, k, \psi]$ supporting point-queries for any potential stream element x (qry_x) and the ability to raise a flag on "bad" frequency estimation. $\mathrm{Qry}_K^{\mathrm{CMS}}, \mathrm{Up}_K^{\mathrm{CMS}}$, resp. $\mathrm{Qry}_K^{\mathrm{HK}}, \mathrm{Up}_K^{\mathrm{HK}}$, denote query and update algorithms of keyed structure CMS$[R, T, m, k]$ (Figure 2), resp. HK$[R, T, m, k, 1]$ (Figure 3). 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, integers $m, k \geq 0$, and flag parameter $\psi \in (0, 1)$. A concrete scheme is given by a particular choice of parameters.

$\Delta \leq (M[i^*][p_{i^*}] - A[i^*][p_{i^*}].\mathrm{cnt} + 1)/2$, 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 now $i^* \in \{j \mid \Theta_2^j = \min_{i \in I_x}\{\Theta_2^i\}\}$. 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 $\mathrm{Qry}(\mathsf{qry}_x)$ 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 $\epsilon = e/m$, $\delta = e^{-k}$.

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.

Nonetheless, we implemented an version of CK with flag-raising (see Figure 15), 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.

References

[1] Bip 37. `https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki`.

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

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

[4] Lada A. Adamic and Bernardo A. Huberman. Zipf's law and the internet. *Glottometrics*, 3(1):143–150, 2002.

[5] 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.

[6] 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.

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

[8] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for

structured data. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. In *AMC SIGCOMM*, 2000.

[15] Mia Filić, Kenny Paterson, Anupama Unnikrishnan, and Fernando Virdia. Adversarial correctness and privacy for probabilistic data structures. In *ACM SIGSAC CCS*, 2022.

[16] 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.

[17] Bob Goodwin, Michael Hopcroft, Dan Luu, Alex Clemmer, Mihaela Curmei, Sameh Elnikety, and Yuxiong He. Bitfunnel: Revisiting signatures for search. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017.

[18] Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In *ACM STOC*, 2013.

[19] Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. In *NeurIPS*, 2020.

[20] Nuno Homem and Joao Paulo Carvalho. Finding top-k elements in data streams. *Information Sciences*, 180(24):4958–4974, 2010.

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

[22] 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.

[23] Bruce M. Maggs and Ramesh K. Sitaraman. Algorithmic nuggets in content delivery. *ACM SIGCOMM CCR*, 45(3):52–66, July 2015.

[24] Ankush Mandal, He Jiang, Anshumali Shrivastava, and Vivek Sarkar. Topkapi: parallel and fast sketches for finding top-k frequent elements. *NeurIPS*, 2018.

[25] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *International Conference on Very Large Databases*, 2002.

[26] Luca Melis, George Danezis, and Emiliano De Cristofaro. Efficient private statistics with succinct sketches. *arXiv preprint arXiv:1508.06110*, 2015.

[27] Herman Melville. *Moby Dick; Or, The Whale.* Project Gutenberg, 1851.

[28] 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.

[29] Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In *Annual Cryptology Conference*, 2015.

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

[31] Raimundo Real and Juan M Vargas. The probabilistic basis of jaccard's index of similarity. *Systematic biology*, 45(3):380–385, 1996.

[32] Tim Roughgarden and Gregory Valiant. Cs168: The modern algorithmic toolbox lecture #2: Approximate heavy hitters and the count-min sketch. page 15.

[33] Benedikt Sigurleifsson, Aravindan Anbarasu, and Karl Kangur. An overview of count-min sketch and its applications. `https://easychair.org/publications/preprint/gNlw`, 2019.

[34] 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.

A Reconsidering the main result of [34]

The main technical result for the HK structure (Theorem 3 in [34]) gives a bound on frequency estimation error in the non-adaptive adversarial setting. However, the assumptions of the theorem statement are under-specified, and the proof itself is incorrect. As a result, the behavior of the HK structure in the presence of non-adaptive attacks remains open. Here we give our own bound, under what we understand to be the assumptions made in [34][Theorem 3].

Fix a target element $x \in \mathcal{U}$, and let $T\colon \mathcal{U} \to \{0,1\}^n$ be the *fingerprint* function for (we omit the key $k \in \mathcal{K}$ for ease of presentation). If some $\tilde{x} \in \mathcal{U}$ is such that $T(\tilde{x}) = T(x)$ and $\tilde{x} \neq x$, then we say that the pair $(\tilde{x}, x)$ constitutes a *fingerprint collision.* Similarly, fix a row $A[j]$ in the HK structure. We say that $(\tilde{x}, x)$ are an h_j*-collision* if $h_j(\tilde{x}) = h_j(x)$ (where h_j is the hash function that maps elements to a particular counter position in any row j of A); note that this implies that $\tilde{x}, x$ both point to the same bin of $A[j]$, and hence they may both affect the counter held there. If x already *owns* that counter, i.e. $A[j][h_j(x)].\mathrm{fp} = T(x)$, then we will refer to any $\tilde{x}$ such that $h_j(\tilde{x}) = h_j(x)$ and $T(\tilde{x}) \neq T(x)$ as an *imposter*.

The following theorem considers a single row $A[j]$ in the HK structure, where a particular counter $A[j][h_j(x)].\mathrm{cnt}$ in that row is owned by some x, i.e., from the viewpoint of that counter x is an "elephant" flow. Any element $\tilde{x}$ that h_j-collides with x will increment the counter if $T(\tilde{x}) = T(x)$, and decrement the counter (with some probability) if $T(\tilde{x}) \neq T(x)$. In the non-adaptive setting, if fingerprints are sufficiently long with respect to the number of distinct elements in the stream, it is unlikely that any $\tilde{x}$ will have the same fingerprint as the *particular* element x — if there are Q distinct elements in the stream, then, with probability at least $1 - Q/2^n$, the fingerprint of x differs from that of all other stream elements. The probability is even lower that any two distinct elements have the same fingerprint and hash to the same counter in any particular row. Under this assumption, any stream element $\tilde{x}$ that h_j-collides with x is an imposter, and may only decrement the counter.

Observe that under the assumption that $\forall \tilde{x} \neq x\colon (h_j(\tilde{x}) = h_j(x)) \Rightarrow T(\tilde{x}) \neq T(x)$, we do not need to distinguish distinct imposter elements; from the point of view of the counter $A[j][h_j(x)].\mathrm{cnt}$ being decremented (or not), all imposters $\tilde{x}$ are the same. Thus, the theorem will assume that there is a single "generic" imposter element $\tilde{x}$, by renaming all imposter elements as $\tilde{x}$. In addition, any stream elements that are not from $\tilde{x}$ nor x are irrelevant to the value of C (a shorthand we will use for the value stored in $A[j][h_j(x)].\mathrm{cnt}$), and can be renamed $\mathtt{Y}$.

Consider a stream of the form $\mathtt{Y}, x, x, \mathtt{Y}, x, \mathtt{Y}, \tilde{x}, x, \mathtt{Y}, \tilde{x}, \tilde{x}, \mathtt{Y}, x, \mathtt{Y}$. First, the packets labeled $\mathtt{Y}$ can be ignored. Now, each occurrence of x will increment C, and each occurrence of $\tilde{x}$ decrements C with a probability that explicitly depends on the value of C at the time that $\tilde{x}$ arrives. In the example, when the first $\tilde{x}$ arrives, $C = 3$, and the probability that this $\tilde{x}$ decrements the counter is d^3; when the second $\tilde{x}$ arrives, the counter value is $C = (3+1) - 1 = 3$ with probability d^3, or $C = (3+1) - 0 = 4$ with probability $1 - d^3$.

This example surfaces a recursive expression. In particular, say there are $n_{\tilde{x}}$ occurrences of $\tilde{x}$ in the stream. For $t = 0, 1, \ldots, n_{\tilde{x}}$, let the t-th *imposter epoch* (or epoch, for short) be defined as the portion of the stream beginning with the t-th occurrence of $\tilde{x}$ and ending one element before the $(t+1)^{\text{th}}$ occurrence of $\tilde{x}$. We make the convention that the 0-th epoch is everything before the first $\tilde{x}$, and the $n_{\tilde{x}}$-th epoch is everything following the last $\tilde{x}$. Let δ_t be the number of packets labeled x in the t-th epoch. (In the above example, $\delta_0 = 3$, $\delta_1 = 1$, $\delta_2 = 0$, $\delta_3 = 1$.) Let C_t be the value of the counter C immediately after the t-th imposter packet is processed, and let $X_t \in \{0, 1\}$ indicate that the t-th imposter packet resulted in the counter being decremented. Then we can write $C_t = C_{t-1} + \delta_t - X_t$, and recursing gives us $C_t = C_0 + \sum_{s=1}^{t} \delta_s - \sum_{s=1}^{t} X_s$. Observe that for each value of t, the sum $n_x^{[t]} = \sum_{s=1}^{t} \delta_s$ counts the number of x packets through the t-th epoch. Then, making the natural conventions that $C_0 = 0, X_0 = 0$, we arrive at the expression $C_t = n_x^{[t]} - \sum_{s=0}^{t} X_s$, or, after rearranging, $n_x^{[t]} - C_t = \sum_{s=0}^{t} X_s$. With this setup in mind, we proceed to give a theorem that bounds the difference between the true count of x, at any point in the stream, and the corresponding value held by $A_j[h_j(x)].\text{cnt}$. We will state the theorem and provide a bit of discussion about what it says, before proceeding to the proof.

Theorem 3. *Fix a element x, and let n_x be the true number of occurrences of x in the stream. Fix a row $j \in \{1, \ldots, k\}$ in an (m, k)-HK structure, and let the decrement parameter be d for some $0 < d \leq 1$. Let h_j be a random mapping from the universe of elements to $\{1, \ldots, m\}$. Assume that:*

1. *no element other than x ever owns the counter $A_j[h_j(x)].\text{cnt}$*
2. *any element $\tilde{x}$ that h_j-collides with x is an imposter (i.e., no fingerprint collisions), and that there are $n_{\tilde{x}} < n_x$ such imposters.*

For $t = 0, \ldots, n_{\tilde{x}}$, let

- *$n_x^{[t]} \leq n_x$ be the number of occurrences of x in the portion of the stream up to the end of the t-th imposter-epoch (i.e., just before the arrival of the $(t+1)$-th imposter),*
- *C_t be the value of $A_j[h_j(x)].\text{cnt}$ at the end of the t-th imposter-epoch (i.e., just before the arrival of the $(t+1)$-th imposter),*
- *$S_t \leq t$ be the number of imposters, among the first t, that cause the counter $A_j[h_j(x)].\text{cnt}$ to decrement.*

Then, for any $v \leq t$,

$$\Pr\left[(n_x^{[t]} - C_t) \leq v\right] > 1 - \frac{1}{v(1 - d^\alpha)}$$

where α is any constant (in particular, the largest one) such that $\forall t > 0$ it holds that $\alpha \leq (n_x^{(t)} - S_{t-1})/t$. $\square$

Before giving the proof, a few remarks. The requirement $\alpha \leq (n_x^{[t]} - S_{t-1})/t$ says, loosely, that across the t "epochs" defined by the arrivals of imposter packets $0, 1, \ldots, t-1$, on average there are at least α packets from x for every *successful* imposter packet, i.e., ones that decrement the counter. Given that x is envisioned to be an "elephant flow", this seems reasonable; in particular, the counter increases as packets from x arrive, and so the probability that the next imposter will be successful decreases exponentially in the counter value. That also suggests that this is a very conservative bound: almost certainly, α is going to be an *increasing* function of the stream length (hence d^α will become quite small, tightening the bound), rather than a constant that must hold for all t.

The theorem assumes that the number of imposters $n_{\tilde{x}} < n_x$, so one might wonder how reasonable is this assumption. Note that the number of imposters $n_{\tilde{x}}$ is a random variable (over the random mapping h_j) whose value is a function of the HK-parameter m (the number of counters in any row) and the number of distinct, non-x elements in the stream. In particular, if the latter is fixed, then $n_{\tilde{x}}$ almost certainly decreases as m increases, and vice versa. Thus, as expected, as m increases there will be fewer opportunities for imposters to decrement the counter $A[j][h_j(x)].\text{cnt}$; in turn, this suggests that α will grow, tightening the bound.

Additionally, recall that an HK point-query for element x is answered with

$$\hat{n}_x = \max_{1 \leq j \leq k} A[j][h_j(x)].\text{cnt}] | A[j][h_j(x)].\text{fp}] = \text{fp}_x,$$

that is the maximum value of the row-counters associated to this element. Observe that during any given imposter-epoch, the difference between the true number of occurrence of x and the counter $A[j][h_j(x)]$.cnt remains *constant*, as both increase by one with each packet from x. In addition, if the hash functions $h_1, \dots, h_k$ are modeled as independent random functions, then the distribution of any row-specific random variable will be the same in every row, and the distribution in any particular row is independent of every other row. Thus, the following corollary.

Corollary 3. *Applying Theorem 3 for each $j \in \{1, \dots, k\}$: at* any *point in the stream, if n_x is the true frequency of x and $\hat{n}_x = HK(x)$, then*

$$\Pr\left[n_x - \hat{n}_x \leq v\right] > \prod_{j=1}^{k} \left(1 - \frac{1}{v(1 - d^{\alpha_j})}\right)$$

where α_j is the constant α from Theorem 3 specific to the j-th row. Conservatively, taking α as the smallest of the α_j, one has $\Pr\left[n_x - \hat{n}_x \leq v\right] > (1 - (1/v(1 - d^{\alpha})))^k$. □

We now proceed to the proof of Theorem 3, with a final note that the proof actually surfaces additional insights, in particular on the outsized effect (on counter "error") of appearances of x early in the stream, versus late in the stream.

Proof of Theorem 3. For $0 \leq t \leq n_{\tilde{x}}$, let X_t be a random variable indicating that the t-th imposter causes a decrement, with the convention that $X_0 = 0$. Clearly, $n_x^{[0]} - C_0 = 0$ as no imposters have yet arrived. Then $\forall 1 \leq i \leq n_{\tilde{x}}, v \leq i$, $\Pr\left[n_x^{[i]} - C_i = v\right] = \Pr\left[\sum_{t=0}^{i} X_t = v\right]$, so bounding $\Pr\left[\sum_{t=0}^{i} X_t = v\right]$ suffices to bound the difference between the true count of x and the estimate provided by the counter. For $i \geq 1$ we can write

$$\begin{aligned}
\Pr\left[\sum_{t=0}^{i} X_t = v\right] &= \Pr\left[X_i = 0 \,|\, \sum_{t=0}^{i-1} X_t = v\right] \Pr\left[\sum_{t=0}^{i-1} X_t = v\right] \\
&\quad + \Pr\left[X_i = 1 \,|\, \sum_{t=0}^{i-1} X_t = v - 1\right] \Pr\left[\sum_{t=0}^{i-1} X_t = v - 1\right] \\
&= \left(1 - d^{n_x^{[i-1]} - v}\right) \Pr\left[\sum_{t=0}^{i-1} X_t = v\right] \\
&\quad + \left(d^{n_x^{[i-1]} - (v-1)}\right) \Pr\left[\sum_{t=0}^{i-1} X_t = v - 1\right]
\end{aligned}$$

and we note that $\Pr\left[\sum_{t=0}^{i-1} X_t = v\right] = 0$ if $v = i$, and $\Pr\left[\sum_{t=0}^{i-1} X_t = v - 1\right] = 0$ if $v = 0$. Recalling the notation that δ_i is the number of x packets in the i-th epoch, this already leads to the simple bound

$$\begin{aligned}
\Pr\left[n_x^{[i]} - C_i \leq (i-1)\right] &= 1 - \Pr\left[n_x^{[i]} - C_i = i\right] \\
&= 1 - d^{(i)\delta_0 + (i-1)\delta_1 + \cdots + 2\delta_{i-2} + \delta_{i-1}) - (i(i-1)/2)}
\end{aligned}$$

which highlights the outsized effect of having a large number of x packets arriving early in the stream, versus late in the stream. If the packets from x arrive at a constant rate, so that $\delta_0 = \delta_1 = \cdots = \delta_{i-1} = r$ this simplifies to

$$\Pr\left[n_x^{[i]} - C_i \leq (i-1)\right] = 1 - d^{\frac{r(i+1)(i)}{2} - \frac{i(i-1)}{2}}.$$

Still, we can say more. Let $S_i = \sum_{t=0}^{i} X_i$ and consider the sequence of random variables $S_0, S_1, \dots, S_i$. We first observe that $S_i - S_{i-1} \leq 1$. Moreover, observe that $\mathbb{E}\left[S_i \,|\, S_0, \dots, S_{i-1}\right] = \mathbb{E}\left[S_i \,|\, S_{i-1}\right]$, and so

$$\begin{aligned}
\mathbb{E}\left[S_i \,|\, S_{i-1}\right] &= \mathbb{E}\left[X_i + S_{i-1} \,|\, S_{i-1}\right] \\
&= \mathbb{E}\left[X_i \,|\, S_{i-1}\right] + S_{i-1} \\
&= d^{n_x^{[i]} - S_{i-1}} + S_{i-1}
\end{aligned}$$

Now using the fact that for any two random variables U, V we have $\mathbb{E}[U] = \mathbb{E}[\mathbb{E}[U \mid V]]$, we have $\mathbb{E}[S_i] = \mathbb{E}[\mathbb{E}[S_i \mid S_{i-1}]] = \mathbb{E}\left[d^{n_x^{[i]} - S_{i-1}}\right] + \mathbb{E}[S_{i-1}]$, which we can telescope out to give $\mathbb{E}[S_i] = \sum_{j=1}^{i} \mathbb{E}\left[d^{n_x^{[j]} - S_{j-1}}\right]$. Recalling that $\mathbb{E}[S_i] = \mathbb{E}\left[\sum_{t=0}^{i} X_t\right]$, by Markov's inequality, we have

$$\Pr\left[n_x^{[i]} - C_i > v\right] \leq \frac{\sum_{j=1}^{i} \mathbb{E}\left[d^{n_x^{[j]} - S_{j-1}}\right]}{v}$$

Let us make the assumption that, for all $j \geq 1$, we have $(1/j)(n_x^{[j]} - S_{j-1}) \geq \alpha$ for some constant $\alpha \geq 1$. Under this assumption, we have $\mathbb{E}\left[d^{n_x^{[j]} - S_{i-1}}\right] \leq d^{\alpha j}$, and $\sum_{j=1}^{i} d^{\alpha j} \leq \sum_{j=1}^{\infty} d^{\alpha j} = 1/(1 - d^{\alpha})$.

Pulling everything together, for all $0 \leq v \leq i$ we have

$$\Pr\left[n_x^{[i]} - C_i \leq v\right] > 1 - \frac{1}{v(1 - d^{\alpha})}$$

$\square$

B More on CMS attacks

The following is a more detailed version of the analysis that appears in Section 5.2. In Figure 5 we give an attack for the private hash and private representation setting. This is the most challenging setting for finding a cover set: the privacy of the hash functions makes local hash computations effectively useless, and the privacy of the representation prevents the adversary from using it to view the result of online hash computations. The attack begins by querying x to learn its current frequency estimate; let $(p_1, p_2, \ldots, p_k) \leftarrow R(K, x)$ and let $M[1][p_1] = c_1, \ldots, M[k][p_k] = c_k$ be the values of the counters associated to x at this time, i.e., $\min_{i \in [k]}\{c_i\} = a \geq 0$.

The attack then inserts distinct random elements that are not equal to x, checking the estimated frequency after each insertion until the estimated frequency for x increases to $a + 1$, as this signals that a cover set for x has been inserted. Let $\vec{I}$ be the stream of inserted elements at the moment that this happens. At this point, we begin the first "round" of extracting from $\vec{I}$ a 1-cover. Say the last inserted element was z_1. As this caused the CMS estimate to increase, z_1 must share at least one counter with x. Moreover, any counter covered by z_1 must have been minimal, i.e., still holding its initial value c_i, at the time that z_1 was inserted. Thus, we set our round-one candidate cover set $\mathcal{C}_1 \leftarrow \{z_1\}$. Notice that by definition, the insertion of z_1 increases the estimation error by one.

Let $\mathcal{M}(\mathcal{C}_1) = \{i \in [k] \mid \exists z \in \mathcal{C} : R(K, z)[i] = p_i\}$, i.e., the set of rows whose x-counters are covered by $\mathcal{C}_1$, and let $\delta_1 = \min_{j \notin \mathcal{M}(\mathcal{C}_1)}\{M[j][p_j]\} - \min_{i \in \mathcal{M}(\mathcal{C}_1)}\{M[i][p_i]\}$. Notice that δ_1 is the gap between the smallest counter(s) *not* covered by $\mathcal{C}_1$, and the smallest counter(s) that are covered by $\mathcal{C}_1$. (Observe that z_1 may also cover non-minimal x-counters.) Thus, if we now reinsert $\mathcal{C}_1$ a total of δ_1 times, this gap shrinks to zero; reinserting it once more will cause some x-counter that is *not* covered by $\mathcal{C}_1$ to become minimal, and we can observe this by making an estimation query (i.e. a **Qry** call) after each reinsertion.

Example: Say we have $k = 4$, and prior to the first insertion of z_1 (as part of $\vec{I}$) we have $M[1][p_1] = 2$, $M[2][p_2] = 3$, $M[3][p_3] = 5$ and $M[4][p_4] = 0$. Now, say that z_1 covers the x-counters in rows 1,4: then upon first inserting z_1, we have $M[1][p_1] = 3$, $M[2][p_2] = 3$, $M[3][p_3] = 5$ and $M[4][p_4] = 1$. We create $\mathcal{C}_1 = \{z_1\}$, and compute $\delta_1 = 3 - 1 = 2$. If we were to insert $\mathcal{C}_1$ twice more, we would have $M[1][p_1] = 5$, $M[2][p_2] = 3$, $M[3][p_3] = 5$ and $M[4][p_4] = 3$; if we had checked the CMS estimate for n_x after each insertion, we would have observed responses 2 and 3. After $\delta_1 + 1 = 3$ re-insertions of $\mathcal{C}_1$, we would have $M[1][p_1] = 6$, $M[2][p_2] = 3$, $M[3][p_3] = 5$, $M[4][p_4] = 4$, and the CMS estimate of n_x would remain 3 because now $M[2][p_2]$ is minimal.$\circ$

Notice that the $\delta_1 + 1$ re-insertions of $\mathcal{C}_1$ will increase the CMS estimate of n_x by exactly δ_1. At this point we begin round 2, searching for $z_2 \in \vec{I} \setminus \mathcal{C}_1$ that covers the newly minimal x-counters. Recall that the elements of $\vec{I}$ are distinct (by design), so if we reinsert $\vec{I} \setminus \mathcal{C}_1$ *in order* we are guaranteed to hit some satisfying $z_2 \neq z_1$, and this can be observed by checking the CMS estimate of n_x after each element is reinserted. As was the case for z_1, we know that z_2 covers the currently minimal x-counters, and that prior to reinserting z_2 these counters had not changed in value since the end of round 1. Thus, reinserting z_2 increases the estimation error by one. We set $\mathcal{C}_2 \leftarrow \mathcal{C}_1 \cup \{z_2\}$, and then switch to reinserting $\mathcal{C}_2$ a total of $\delta_2 + 1$ times (where δ_2 is defined analogously to δ_1) to end round 2. Again, this increases the estimation error by δ_2.

Continuing this way, after some $\ell \leq k$ rounds we will have found a complete 1-cover for x. There can be at most k rounds, because each round i adds exactly one new element z_i to the incomplete cover $\mathcal{C}_{i-1}$, and there are only k counters to cover. Notice that in round ℓ, when we reinsert $\mathcal{C}_\ell$ we will never observe that some new x-counter has become minimal: all x-counters are covered by $\mathcal{C}_\ell$, so all will be increased by each reinsertion. Nonetheless, each reinsertion of $\mathcal{C}_\ell$ adds one to the estimation error, and these re-insertions may continue until the resource budget is exhausted, i.e., until a total of q_U elements have been inserted (via **Up**) as part of the attack.

The number of **Up**-queries (i.e. insertions) required to reach the complete cover $\mathcal{C}_\ell$ is

$$q'_U \leq \ell|\vec{I}| + \sum_{i=1}^{\ell-1}(\delta_i + 1)(i) = \ell|\vec{I}| + \frac{\ell(\ell-1)}{2} + \sum_{i=1}^{\ell-1} i\delta_i$$

and so $\mathcal{C}_\ell$ can potentially be reinserted at least $\lfloor (q_U - q'_U)/\ell \rfloor$ times, each time adding one to the estimation error. We say *potentially* because the **Qry**-query budget may be the limiting factor; we'll return to this in a moment. For now, assuming q_Q is not the limiting factor, the error introduced by the attack is

$$\begin{aligned} \mathsf{Err} &\geq \left\lfloor \left(\ell + \sum_{i=1}^{\ell-1}\delta_i\right) + \left(\frac{q_U - \ell|\vec{I}| - \frac{\ell(\ell-1)}{2} - \sum_{i=1}^{\ell-1} i\delta_i}{\ell}\right)\right\rfloor \\ &= \left\lfloor \left(\frac{\ell+1}{2} + \frac{1}{\ell}\left(q_U + \sum_{i=1}^{\ell-1}(\ell-i)\delta_i\right) - L^1\right)\right\rfloor \end{aligned}$$

where the final line holds because $|\vec{I}|$ is, by construction, precisely L_1. We note that Err is a function of several random variables: $L^1, \ell, \{\delta_i\}_{i\in[\ell-1]}$.

We would like to develop an expression for $\mathbb{E}[\mathsf{Err}]$, so we observe that for practical values of k, m (e.g., $k = 4$, with $m \gg k$) it is likely that $\ell = k$. We have $\ell < k$ only if one or more of the covering elements cover multiple x-counters, and for small $k \ll m$ this is unlikely. We approximate Err with $\widehat{\mathsf{Err}}$ by replacing ℓ with k, dropping the flooring operation, arriving at

$$\mathbb{E}[\mathsf{Err}] \approx \mathbb{E}[\widehat{\mathsf{Err}}] \approx \left(\frac{k+1}{2} + \frac{1}{k}\left(q_U + \sum_{i=1}^{k-1}(k-i)\mathbb{E}[\delta_i]\right) - \mathbb{E}[L^1]\right)$$

Rearranging and using the very tight approximation $\mathbb{E}[L^1] \approx mH_k$, we have

$$\mathbb{E}[\widehat{\mathsf{Err}}] \approx \left(\frac{q_U}{k} - mH_k\right) + \frac{k+1}{2} + \left(\frac{1}{k}\sum_{i=1}^{k-1}(k-i)\mathbb{E}[\delta_i]\right)$$

We do not have a crisp way to describe the distribution of the δ_i random variables, but we can make some educated statements about them. The expected value of *any* counter $M[i][j]$ after $\vec{I}$ has been inserted is $|\vec{I}|/m \approx mH_k/m = H_k$, and $H_k < 4$ for $k \leq 30$ (and practical values of k are typically much less than 30); moverover, standard balls-and-bins arguments tell us that as the number of balls approaches $m \ln m$, the *maximum* counter value in any row approaches the expected value. Since $\delta_1 \leq \max_{j\notin\mathcal{M}(\{z_1\})}\{M[j][p_j]\} - \min_{i\in\mathcal{M}(\{z_1\})}\{M[i][p_i]\}$, we can safely assume that $\mathbb{E}[\delta_1]$ is upper-bounded by a constant that is small relative to $m, q_U/k$.

After inserting $\mathcal{C}_1 = \{z_1\}$ a total of $\delta_1 + 1$ times, we switch to reinserting $\vec{I} \setminus \mathcal{C}_1$ until we find a z_2 that covers the currently minimal x-counters. When we begin to reinsert $\mathcal{C}_2$, we know by construction that

$\delta_2 \leq \min_{j \notin \mathcal{M}(\{z_1,z_2\})}\{M[j][p_j]\} - \big(\min_{j \notin \mathcal{M}(\{z_1\})}\{M[j][p_j]\} + 1\big)$. For the first term in the difference, we "roll back" one round; say that $\alpha = \max_{j \notin \mathcal{M}(\{z_1\})}\{M[j][p_j]\}$. Then, being very pessimistic, we know that $\min_{j \notin \mathcal{M}(\{z_1,z_2\})}\{M[j][p_j]\} \leq 2\alpha + (\delta_1 + 1)$: in finding z_2, we reinsert at most all of $\vec{I} \setminus \{z_1\}$, which would add another (at most) α to that maximum counter value, and the repeated insertions of $\mathcal{C}_1$ could have added at most $\delta_1 + 1$ to said maximum counter. However, the second term in the difference is at least $\delta_1 + 1$, so $\delta_2 \leq 2\alpha$ and we have already argued that α is in the neighborhood of $H_k < 4$. Continuing this this way, we reach the conclusion that 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 in Table 2. For realistic values of k, significant error will be created when $q_U \gg (mk)H_k$. For example, when $k = 4, m = 2048$ we require $q_U \gg 17067$; this is likely not a real restriction in most practical use-cases of CMS, e.g., computing the heavy hitter flows traversing a router.

Returning to the matter of exhausting the **Qry**-budget, the total number of **Qry**-queries for the attack depends somewhat heavily on whether or not $\ell = k$. If $\ell = k$ then $|\mathcal{C}_k| = k$, and we know that a complete cover has been found. Thus, we do not need to make any **Qry**-queries during reinsertions of $\mathcal{C}_k$. If $\ell < k$, however, then we must make **Qry**-queries during reinsertions of $\mathcal{C}_\ell$, because we do not know that $\mathcal{C}_\ell$ contains a complete cover.

Either way, the number of **Qry**-queries need to reach $\mathcal{C}_\ell$ is $q'_Q \leq 1 + \ell|\vec{I}| + \sum_{i=1}^{\ell-1}(\delta_i + 1)$, and the expected gap between q'_U and q'_Q is

$$\begin{aligned}
\mathbb{E}[q'_U - q'_Q] &\approx \mathbb{E}\left[\sum_{i=1}^{\ell-1} i(\delta_i + 1) - \sum_{i=1}^{\ell-1}(\delta_i + 1)\right] \\
&\leq \mathbb{E}\left[\sum_{i=1}^{\ell-1} \delta_i\right] + \frac{(k-1)(k-2)}{2} \\
&\leq k\mathbb{E}\left[\max_{i \in [\ell-1]}\{\delta_i\}\right] + \frac{(k-1)(k-2)}{2}
\end{aligned}$$

By the arguments just given about the δ_i, we can safely bound $\mathbb{E}\left[\max_{i \in [\ell-1]}\{\delta_i\}\right]$ by kH_k. So $\mathbb{E}[q'_U - q'_Q] = O(k^2)$ with a small hidden constant. Thus, the expected numbers of **Up**-queries and **Qry**-queries expended to find the complete cover $\mathcal{C}_\ell$ are similar, especially for realistic values of k.

Now, in the most likely case that $\ell = k$, no further **Qry**-queries are needed. Hence, when $\ell = k$, the overall error induced by the attack will be determined by the insertion/**Up**-budget (q_U) when the *total* **Qry**-budget q_Q is approximately the insertion-budget required for finding the cover. When $\ell < k$, in order for the overall error to be determined by the insertion budget, the total **Qry**-budget needs to accommodate $q'_Q + (q_U - q'_U)/\ell$ queries. The second summation comes from the fact that while accumulating error via re-insertions of $\mathcal{C}_\ell$, we must make one **Qry**-query per reinsertion. This is a potentially large jump in the number of estimation queries required, from $\ell = k$ to $\ell < k$. But in reality the jump might be less important than it appears: if $\ell < k$ then given our intuition about the δ_i, it seems likely that if some $\mathcal{C}_i$ is taking a large number of insertions, one can likely assume that $\mathcal{C}_i$ is a complete cover, cease making estimation queries and switch to an insertion only strategy.

C Delayed Proofs for CK results

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 $n_x \leq \frac{M[i][p_i] - A[i][p_i].\mathrm{cnt} + 1}{2} = \Theta_1^i$.* ♦

Proof of Lemma 1. We can think of the counter $A[i][p_i].\mathrm{cnt}$ as counting the depth of a stack of fingerprint-labeled plates. The rules of the stack are as follows. Upon insertion of x into the CK structure:

1. if $A[i][p_i].\mathrm{cnt} = 0$ then the stack is empty; then push an fp_x-labeled plate and set $A[i][p_i].\mathrm{cnt} \leftarrow 1, A[i][p_i].\mathrm{fp} \leftarrow \mathrm{fp}_x$.

2(a). if $A[i][p_i].\text{cnt} = c > 0$ and $A[i][p_i].\text{fp} = \text{fp}_x$, then push an fp_x-labeled plate on to the stack and increment $A[i][p_i].\text{cnt} \leftarrow c+1$.

2(b). if $A[i][p_i].\text{cnt} = c > 0$ and $A[i][p_i].\text{fp} \neq \text{fp}_x$, then pop the top (fp-labeled) plate and decrement $A[i][p_i].\text{cnt} \leftarrow c-1$. If this causes $A[i][p_i].\text{cnt} = 0$, then push an fp_x-labeled plate and set $A[i][p_i].\text{cnt} \leftarrow 1, A[i][p_i].\text{fp} \leftarrow \text{fp}_x$.

These stack rules are precisely the CK rules for handling insertions. Now, upon the first insertion to CK, by rule 1 it is clear that all plates on the stack (there is only one of them) have label $A[i][p_i].\text{fp}$, and $A[i][p_i].\text{cnt}$ is the number (1) of plates on the stack. Inductively, assume that $A[i][p_i].\text{cnt} = c > 0$ and all c of the plates on the stack have the same label $A[i][p_i].\text{fp}$. Say that the next insertion is z and $A[i][p_i].\text{fp} = \text{fp}_z$. By rule 2(a), we push an fp_z-plate on to the stack and increment $A[i][p_i].\text{cnt} \leftarrow c+1$. In this case, by assumption, it remains the case that all plates have the same label equal to $A[i][p_i].\text{fp}$, and there are $c+1$ of them. Alternatively, if $A[i][p_i].\text{fp} \neq \text{fp}_z$ then by rule 2(b) we pop the top plate and decrement $A[i][p_i].\text{cnt} \leftarrow c-1$. At this point, either the stack is empty and $A[i][p_i].\text{cnt} = 0$, so by 2(b) we push an fp_z-plate and set $A[i][p_i].\text{cnt} \leftarrow 1$ and $A[i][p_i].\text{fp} \leftarrow \text{fp}_z$; or the stack is not empty, and we take no further action. In the first case, the stack contains a single plate labeled with $A[i][p_i].\text{fp}$ and the counter is 1; in the second, by assumption all plates on the stack are still labeled with $A[i][p_i].\text{fp}$, and $A[i][p_i].\text{cnt}$ still gives the number of plates on the stack.

Having shown the invariant of the stack, we make the following observation. Let $\tilde{n} = \sum_{y \in V_x^i} n_y$. Then $M[i][p_i] = n_x + \tilde{n}$. By the statement of the lemma $i \in \hat{I}_x$, implying that $A[i][p_i].\text{fp} \neq \text{fp}_x$. We claim that $A[i][p_i].\text{cnt} = c > 0$ implies $\tilde{n} - n_x \geq A[i][p_i].\text{cnt} - 1$. To see this, note that $A[i][p_i].\text{cnt} = c > 0$ means that there are c plates labeled with $A[i][p_i].\text{fp} \neq \text{fp}_x$ on the stack associated to c insertions of elements in V_x^i with fingerprint $A[i][p_i].\text{fp}$. If there ever were any fp_x-labeled plates on the stack (i.e., $n_x > 0$), they were subsequently popped off by insertions of elements with their fingerprints not equal to fp_x. On the other hand, if an insertion of x did not place a plate on to the stack, then it popped off a plate corresponding to an insertion of an element in V_x^i. Thus, at most $\tilde{n} - n_x$ insertions of elements in V_x^i have never popped off a plate of x, or had their plate popped off by an insertion of x. For $\tilde{n} - n_x = 0$ we have that $A[i][p_i].\text{cnt} = 1$, and $\tilde{n} - n_x \geq A[i][p_i].\text{cnt} - 1$. Similarly, if $\tilde{n} - n_x = d > 0$ then $\tilde{n} - n_x \geq A[i][p_i].\text{cnt} - 1$ as there are still $A[i][p_i].\text{cnt}$ plates associated with insertions of elements in V_x^i that have never been popped off and at least $A[i][p_i].\text{cnt} - 1$ of them correspond to insertions not popping off a plate of x.

We conclude that $M[i][p_i] = n_x + \tilde{n} \geq n_x + (n_x + A[i][p_i].\text{cnt} - 1) = 2n_x + A[i][p_i].\text{cnt} - 1$. Or, by rearranging,

$$n_x \leq \frac{M[i][p_i] - A[i][p_i].\text{cnt} + 1}{2}$$

which proves the lemma. □

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

Proof of Lemma 2. We can think of the counter $A[i][p_i].\text{cnt}$ as counting the depth of a stack of fingerprint-labeled plates as for the proof of Lemma 1. View an insertion of x being associated with either an insertion of $y \in V_x^i$ that pops off its fp_x-labelled plate from the stack or an insertion of $y \in V_x^i$ of the plate it pops off.

By the statement of the lemma $i \in I_x$, $A[i][p_i].\text{fp} = \text{fp}_x$ and under the NFC condition all plates on the stack are of x. Out of the insertions having plates on the stack, only the bottom plate one could have popped off a plate of $y \in V_x^i$. Thus, at least $n_x - A[i][p_i].\text{cnt}$ insertions of x are associated with an (unique) insertion of $y \in V_x^i$ and

$$\tilde{n} \geq n_x - A[i][p_i].\text{cnt}.$$

From $M[i][p_i] = n_x + \tilde{n}$ we thus obtain $M[i][p_i] \geq 2n_x - A[i][p_i].\text{cnt}$ and

$$n_x \leq \frac{M[i][p_i] + A[i][p_i].\text{cnt}}{2}.$$

□

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

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

Proof of corollary 1. We think of the counter $A[i][p_i].\mathrm{cnt}$ as counting the depth of a stack of fingerprint-labeled plates as for the proof of Lemma 1 and associate occurrences of x and $y \in V_x^i$ in the similar way. Moreover, $|V_x^i| = 1$ implies $M[i][p_i] = n_x + n_z$, or equivalently, $n_z = M[i][p_i] - n_x$ for $z \neq x$.

We start by focusing on the case $i \in \hat{I}_x$ ($A[i][p_i].\mathrm{fp} \neq \mathrm{fp}_z$). Say x at some point owned the counter. Then, the plate at the bottom of the stack (labeled with fp_z) corresponds to a occurrence of z that popped off a plate of x. So, only $A[i][p_i].\mathrm{cnt} - 1$ occurrences of z are not associated with x implying $n_z = n_x + A[i][p_i].\mathrm{cnt} - 1$. Hence, $M[i][p_i] - n_x = n_x + A[i][p_i].\mathrm{cnt} - 1$, or equivalently, $n_x = \frac{M[i][p_i] - A[i][p_i].\mathrm{cnt} + 1}{2}$. Say x never owned the counter. Then, none of the occurrences of z with a plate on the stack popped an x-plate from the stack. This implies that $n_z = n_x + A[i][p_i].\mathrm{cnt}$, and $n_x = \frac{M[i][p_i] - A[i][p_i].\mathrm{cnt}}{2}$.

Let now $i \in I_x$ ($A[i][p_i].\mathrm{fp} = \mathrm{fp}_x$). Say x was the only owner of the counter. Then, none of the occurrences of x with a plate on the stack popped an z-plate from the stack. Thus, $n_x = n_z + A[i][p_i].\mathrm{cnt}$ and, adding n_x to both sides and rearranging, $n_x = \frac{M[i][p_i] + A[i][p_i].\mathrm{cnt}}{2}$. Say z at some point owned the counter. Then, the plate at the bottom of the stack (labeled with fp_x) corresponds to the occurrence of x that popped off a plate of z, and $n_x = n_z + A[i][p_i].\mathrm{cnt} - 1$ and $n_x = \frac{M[i][p_i] + A[i][p_i].\mathrm{cnt} - 1}{2}$. $\square$

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

Proof of corollary 2. The NFC condition gives $\mathrm{CK}(x) \geq n_x \geq \mathrm{HK}(x)$, and $\mathrm{CK}(x) - n_x \leq \mathrm{CK}(x) - \mathrm{HK}(x)$. So, by Lemma 3 we arrive at

$$\begin{aligned}
\mathrm{CK}(x) - n_x &\leq \left(\frac{\mathrm{CMS}(x) + \mathrm{HK}(x)}{2}\right) - n_x \\
&\leq \left(\frac{\mathrm{CMS}(x) + \mathrm{HK}(x)}{2}\right) - \mathrm{HK}(x) \\
&\leq \frac{\mathrm{CMS}(x) - \mathrm{HK}(x)}{2}.
\end{aligned}$$

$\square$