

DATA STRUCTURES IN ADVERSARIAL ENVIRONMENTS

By

SAM A. MARKELON

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL  
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT  
OF THE REQUIREMENTS FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2025

© 2025 Sam A. Markelon

For my dearest Julia, without whom this would mean less

## ACKNOWLEDGEMENTS

- Family – Dad, Mom, Jack, Hannah, Nonni and Boppi, Kelly and Brian; plus extended – mention upbringing in Burlington, CT
- UConn mentors – Joo, Krawec, Fuller, Herzberg – others?
- Tom
- Committee – VB, SR, PT, JB
- FICS lab – KB, students past and present
- ETHZ Applied Cryptography group: Kenny, Mia, Nico – more by name?
- TUD Cryptoplexity Group – Marc and Moritz
- Friends – Gainesville (Tim, JT, Jefferson, Logan, Gavin, Trivia Group), CT (all in Rod Stewart Chat), all those I forgot to name.
- Julia

## TABLE OF CONTENTS

	<u>page</u>
ACKNOWLEDGEMENTS .....	4
LIST OF TABLES.....	7
LIST OF FIGURES.....	8
ABSTRACT.....	9
CHAPTER	
1 INTRODUCTION .....	10
2 BACKGROUND.....	16
2.1 Notation.....	16
2.2 A Syntax for Data Structures .....	17
2.3 Streaming Data .....	19
3 COMPACT FREQUENCY ESTIMATORS IN ADVERSARIAL ENVIRONMENTS.....	20
3.1 Formal Attack Model .....	20
3.2 Count-min Sketch.....	22
3.3 HeavyKeeper .....	23
3.4 Attacks on CMS and HK .....	25
3.4.1 Cover Sets .....	26
3.4.2 Cover-Set Attacks on CMS.....	28
3.4.3 Cover-Set Attacks on HK .....	37
3.5 Count-Keeper.....	45
3.5.1 Structure.....	46
3.5.2 Correcting CMS and Correctness of CK.....	48
3.5.3 Frequency estimate errors.....	52
3.5.4 Experimental Results .....	54
3.5.5 Attacks Against the CK.....	62
3.5.6 Adversarial Robustness .....	68
4 PROBABILISTIC DATA STRUCTURES IN THE WILD: A SECURITY ANALYSIS OF REDIS .....	71
5 PROVABLY ROBUST SKIPPING-BASED PROBABILISTIC DATA STRUCTURES ...	72
6 CONCLUSION AND FUTURE WORK .....	73

LIST OF REFERENCES.....	74
BIOGRAPHICAL SKETCH .....	77

## LIST OF TABLES

<u>Tables</u>	<u>page</u>
3-1 Non-adversarial CFE Results. ....	60
3-2 CFE Attack Comparison.....	66

## LIST OF FIGURES

<u>Figures</u>	<u>page</u>
3-1 The ERR-FE Attack Model. ....	21
3-2 The Count-min Sketch Structure. ....	22
3-3 The HeavyKeeper Structure. ....	24
3-4 Public Hash CMS Attack. ....	30
3-5 Private Hash and Private Representation CMS Attack. ....	31
3-6 Private Hash and Public Representation CMS Attack.....	37
3-7 Public Hash HK Attack.....	40
3-8 Private Hash and Private Representation HK Attack.....	41
3-9 Private Hash and Public Representation Attack. ....	44
3-10 The Count-Keeper Structure. ....	47
3-11 Stream Frequent Elements.....	56
3-12 Public Hash CK Attack.....	64
3-13 Private Hash and Private Representation CK Attack.....	65
3-14 Private Hash and Public Representation CK Attack. ....	67
3-15 Robust Flag Raising Count-Keeper Structure. ....	70



Abstract of Dissertation Presented to the Graduate School  
of the University of Florida in Partial Fulfillment of the  
Requirements for the Degree of Doctor of Philosophy

DATA STRUCTURES IN ADVERSARIAL ENVIRONMENTS

By

Sam A. Markelon

August 2025

Chair: Vincent Bindschaedler

Major: Computer Science

This is the abstract tex file, which should have been set in the main file using the command

`\setAbstractFile{Drive:/file/location/abstractFile}.`

This is what will appear in the place of an abstract, no formatting or other content is needed, just fill this file with your actual abstract, eg; In this paper we give examples of the various files and configurations used in the graduate school L<sup>A</sup>T<sub>E</sub>X template for dissertations and thesis papers. It should be 350 words or less.

## CHAPTER 1 INTRODUCTION

**The following needs to be re-organized and re-written.**

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.

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 [1] admits data-membership queries (*Does element  $x$  appear in the data?*). Bloom filters are used in applications such as increasing cache performance [2], augmenting the performance of database queries [3], indexing search results [4], and Bitcoin wallet synchronization [5]. 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 [6], Clayton, Patton and Shrimpton [7], and Filić et al. [8] — have provided detailed analyses of Bloom filters under *adaptive* attacks; the results are overwhelmingly negative. Paterson and Raynal [9] provided similar results for the HyperLogLog PDS, which can be used to count the number of distinct elements in a data collection [10].

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 [11], network monitoring [12], recommendation systems [13], and approximate database queries [14], to name a few applications.

The Count-min Sketch (CMS) [15] and HeavyKeeper (HK) [16] 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. [17]. The HK structure is the CFE of choice in the RedisBloom module [14], a component of the Redis database system [18].

Of particular interest to us is the 2019 ACM SIGSAC work of Clayton, Patton, and Shrimpton [7] 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  $V_x^i = \{y \in \vec{S} \mid y \neq x \text{ and } h_i(y) = p_i\}$  be the set of elements that hash to the same counter as  $x$ , in

the  $i$ -th row. Then we can write  $M[i][p_i] = n_x + \sum_{y \in V_x^i} n_y$ , where the  $n_y > 0$  are the true frequencies of the colliding ys. Viewed this way, we see that the CMS estimate  $\hat{n}_x$  minimizes the impact of “collision noise”, i.e.,  $\hat{n}_x = n_x + \min_{i \in [k]} \{\sum_{y \in V_x^i} n_y\}$ .

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

### **The Count-Keeper CFE.**

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

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

As a side note, we uncovered numerous analytical errors in [16] that invalidate some of their claims about the behaviors of the HK structure. We have communicated with the authors of [16] 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 [20], Cohen et al. [21] and Ben-Eliezer et al. [22] 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.

## CHAPTER 2 BACKGROUND

**The following needs to be re-organized and re-written.**

### 2.1 Notation

#### Bitstring and Set Operations.

Let  $\{0, 1\}^*$  denote the set of bitstrings and let  $\varepsilon$  denote the empty string. Let  $X \parallel Y$  denote the concatenation of bitstrings  $X$  and  $Y$ . When  $S$  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  $S$ , we overload the  $\parallel$  operator and write  $S \parallel e$ .

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

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

#### Functions.

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

#### Arrays and Tuples.

We use the distinguished symbol  $\star$  to mean that a variable is uninitialized. By  $[\text{item}] \times \ell$  for  $\ell \in \mathbb{N}$  we mean a vector of  $\ell$  replicas of item. We use  $\text{zeros}(m)$  denote a function that returns an  $m$ -length array of 0s and, likewise,  $\text{zeros}(k, m)$  to denote a function that returns an  $k \times m$  array of



0s. We index into arrays (and tuples) using  $[\cdot]$  notation; in particular, if  $R$  is a function returning a  $k$ -tuple, we write  $R(x)[i]$  to mean the  $i$ -th element/coordinate of  $R(x)$ . If  $X = (x_1, x_2, \dots, x_t)$  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.

## 2.2 A Syntax for Data Structures

We present (a slightly modified) syntax for data structures first provided by [7]. While originally used to describe a variety of probabilistic data structures, the syntax is appropriately general. A syntactic formalization of data structures in this way not only allows us to elegantly describe numerous data structures, but also craft security definitions that are directly related to the operations the data structure allows. We will do exactly this in our case studies throughout the rest of this work.

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

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

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

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

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

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

### 2.3 Streaming Data

A *stream* data-object  $\vec{S} = e_1, e_2, \dots$  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.

## CHAPTER 3

### COMPACT FREQUENCY ESTIMATORS IN ADVERSARIAL ENVIRONMENTS

#### 3.1 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 3-1. The experiment parameters  $u, v$  determine whether the adversary  $\mathcal{A}$  is given  $K$  and  $\text{repr}$ , 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 **Up**( $e$ ) to mean an insertion of  $e$  into the structure and **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 **Hash**:  $\mathcal{X} \rightarrow \mathcal{Y}$ , 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 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  $\mathbf{Atk}_{\Pi}^{\text{err-fe}[u,v]}(\cdot)$  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.

$\text{Atk}_{\Pi, \mathcal{U}}^{\text{err-fe}[u,v]}(\mathcal{A})$	$\text{Up}(\text{up})$
1 : $\vec{S} \leftarrow \emptyset; K \leftarrow \mathcal{K}$	1 : $\text{repr}' \leftarrow \text{UP}_K(\text{repr}, \text{up})$
2 : $\text{repr} \leftarrow \text{REP}_K(\vec{S})$	2 : $\vec{S} \leftarrow \text{up}(\vec{S})$
3 : $\text{kv} \leftarrow \top; \text{rv} \leftarrow \top$	3 : $\text{repr} \leftarrow \text{repr}'$
4 : <b>if</b> $u = 1$ : $\text{kv} \leftarrow K$	4 : <b>if</b> $v = 0$ : <b>return</b> $\top$
5 : <b>if</b> $v = 1$ : $\text{rv} \leftarrow \text{repr}$	5 : <b>return</b> $\text{repr}$
6 : $x \leftarrow \mathcal{U}$	<b>Qry</b> (qry)
7 : $\text{done} \leftarrow \mathcal{A}^{\text{Hash}, \text{Up}, \text{Qry}}(x, \text{kv}, \text{rv})$	1 : <b>return</b> $\text{QRY}_K(\text{repr}, \text{qry})$
8 : $n_x \leftarrow \text{qry}_x(\vec{S})$	<b>Hash</b> (X)
9 : $\hat{n}_x \leftarrow \text{QRY}_K(\text{repr}, \text{qry}_x)$	1 : <b>if</b> $X \notin \mathcal{X}$ : <b>return</b> $\perp$
10 : <b>return</b> $ \hat{n}_x - n_x $	2 : <b>if</b> $H[X] = \perp$
	3 : $H[X] \leftarrow \mathcal{Y}$
	4 : <b>return</b> $H[X]$

Figure 3-1. 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**.

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.

### 3.2 Count-min Sketch

$\text{REP}_K(\mathcal{S})$	$\text{UP}_K(M, \text{up}_x)$
1 : $M \leftarrow \text{zeros}(k, m)$	1 : $(p_1, \dots, p_k) \leftarrow R(K, x)$
2 : <b>for</b> $x \in \mathcal{S}$	2 : <b>for</b> $i \in [k]$
3 : $M \leftarrow \text{UP}_K(M, \text{up}_x)$	3 : $M[i][p_i] += 1$
4 : <b>return</b> $M$	4 : <b>return</b> $M$
	$\text{QRY}_K(M, \text{qry}_x)$
	1 : $(p_1, \dots, p_k) \leftarrow R(K, x)$
	2 : <b>return</b> $\min_{i \in [k]} \{M[i][p_i]\}$

Figure 3-2. Keyed count-min sketch structure  $\text{CMS}[R, m, k]$  admitting point queries for any  $x \in \mathcal{U}$ . The parameters are integers  $m, k \geq 0$ , and a keyed function  $R : \mathcal{K} \times \mathcal{U} \rightarrow [m]^k$  that maps data-object elements (encoded as strings) to a vector of positions in the array  $M$ . A concrete scheme is given by a particular choice of parameters.

Figure 3-2 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  $R(K, x) = (p_1, p_2, \dots, p_k)$ , and then adding 1 to each of the counters at  $M[i][p_i]$ . Traditionally, it is assumed that  $(p_1, \dots, p_k) = (h_1(x), \dots, h_k(x))$  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  $\text{QRY}(\text{qry}_x)$  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  $V_x^i = \{y \in \vec{S} \mid y \neq x \text{ and } R(y)[i] = p_i\}$  be the set of elements that “collide” with  $x$ ’s counter in the  $i$ -th row. Then we can write

$M[i][p_i] = n_x + \sum_{y \in V_x^i} n_y$ , where  $n_y \geq 0$ . Viewed this way, we see that a CMS estimate  $\hat{n}_x$  minimizes the “collision noise”, i.e.,  $\hat{n}_x = n_x + \min_{i \in [k]} \{\sum_{y \in V_x^i} n_y\}$ .

For any  $\epsilon, \delta \geq 0$ , any  $x \in \mathcal{U}$ , and any stream  $\vec{S}$  (over  $\mathcal{U}$ ) of length  $N$ , it is guaranteed that  $\Pr[\hat{n}_x - n_x > \epsilon N] \leq \delta$  when: (1)  $k = \lceil \ln \frac{1}{\delta} \rceil$ ,  $m = \lceil \frac{N}{\epsilon} \rceil$ , and (2)  $R(K, x) = (h_1(K \parallel x), h_2(K \parallel x), \dots, h_k(K \parallel x))$  for  $h_i$  that are uniformly sampled from a pairwise-independent hash family  $H$  [15]. 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.

### 3.3 HeavyKeeper

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

$\text{REP}_K(\mathcal{S})$	$\text{UP}_K(A, \text{up}_x)$
1 : / initialise $k \times m$ (fp,cnt) 2-d array 2 : <b>for</b> $i \in [k]$ 3 : $A[i] \leftarrow [(\star, 0)] \times m$ 4 : <b>for</b> $x \in \mathcal{S}$ 5 : $A \leftarrow \text{UP}_K(A, \text{up}_x)$ 6 : <b>return</b> $A$	1 : $(p_1, \dots, p_k) \leftarrow R(K, x)$ 2 : $\text{fp}_x \leftarrow T(K, x)$ 3 : <b>for</b> $i \in [k]$ 4 : <b>if</b> $A[i][p_i].\text{fp} \notin \{\text{fp}_x, \star\}$ 5 : $r \leftarrow [0, 1)$ 6 : <b>if</b> $r \leq d^{A[i][p_i].\text{cnt}}$ 7 : $A[i][p_i].\text{cnt} -= 1$ 8 :   / overtake the counter if 0 9 : <b>if</b> $A[i][p_i].\text{cnt} = 0$ 10 : $A[i][p_i].\text{fp} \leftarrow \text{fp}_x$ 11 :   / increase the count if fp = $\text{fp}_x$ 12 : <b>if</b> $A[i][p_i].\text{fp} = \text{fp}_x$ 13 : $A[i][p_i].\text{cnt} += 1$ 14 : <b>return</b> $A$
$\text{QRY}_K(A, \text{qry}_x)$	
1 : $(p_1, \dots, p_k) \leftarrow R(K, x)$ 2 : $\text{fp}_x \leftarrow T(K, x)$ 3 : $\text{cnt}_x \leftarrow 0$ 4 : <b>for</b> $i \in [k]$ 5 : <b>if</b> $A[i][p_i].\text{fp} = \text{fp}_x$ 6 : $\text{cnt} \leftarrow A[i][p_i].\text{cnt}$ 7 : $\text{cnt}_x \leftarrow \max\{\text{cnt}_x, \text{cnt}\}$ 8 : <b>return</b> $\text{cnt}_x$	

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



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

The HK provides frequency estimates via point-queries. Writing  $(p_1, \dots, p_k) \leftarrow R(K, x)$  and  $\text{fp}_x \leftarrow T(K, x)$ , a point-query for  $x$  returns  $\max \{A[i][p_i].\text{cnt} \mid A[i][p_i].\text{fp} = \text{fp}_x, i \in [k]\}$ , 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.[16] do state a probabilistic guarantee on the size of estimation errors, under an assumption that each  $A[i][j]$  has one and only one owner for the duration of the stream, but the statement is insufficiently precise and its proof is flawed, so we will not quote it. In the full version of our paper [23], we recover a meaningful result (under their assumptions).

### 3.4 Attacks on CMS and HK

In the following discussion of attacks against CMS and HK in our formal model, we will implement the mappings  $R : \mathcal{U} \rightarrow [m]^k$  and  $T : \mathcal{K} \times \{0, 1\}^* \rightarrow \{0, 1\}^n$  via calls to the **Hash**-oracle. In detail, given some unambiguous encoding function  $\langle \cdot, \cdot, \cdot \rangle$ , for CMS we set  $R(K, x) = (\mathbf{Hash}(\langle 1, K, x \rangle), \mathbf{Hash}(\langle 2, K, x \rangle), \dots, \mathbf{Hash}(\langle k, K, x \rangle))$ , 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  $\mathcal{U} \rightarrow [m]$ . 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”.<sup>1</sup>

### 3.4.1 Cover Sets

Say  $\hat{n}_x$  is the CMS estimate. As noted in Section 3.2, 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  $V_x^1, \dots, V_x^k$  are all non-empty. When this is the case, the union  $C = \bigcup_{i \in [k]} V_x^i$  contains a set of stream elements that “cover” the counters  $M[i][p_i]$  associated to  $x$ . Since the presence of a covering  $C$  within the stream is necessary (and sufficient) for creating a frequency estimation error for the CMS, we formalize the idea of a “cover” in the following definition.

**Definition 1.** Let  $\mathcal{U}$  be the universe of possible stream elements. Fix  $x \in \mathcal{U}$ ,  $r \in \mathbb{Z}$ , and  $\mathcal{Y} \subseteq \mathcal{U}$ .

Then a set  $C = \{y_1, y_2, \dots, y_t\}$  is an  $(\mathcal{Y}, x, r)$ -cover if: (1)  $C \subseteq \mathcal{Y} \setminus \{x\}$ , and (2)  $\forall i \in [k]$

$\exists j_1, \dots, j_r \in [t]$  such that  $R(K, x)[i] = R(K, y_{j_1})[i], \dots, R(K, x)[i] = R(K, y_{j_r})[i]$ .  $\blacklozenge$

For the CMS, we will be interested in  $\mathcal{Y} = \mathcal{U}$ ,  $r = 1$ , and we will shorten the notation to calling this a 1-cover (for  $x$ ), or just a cover. For the HK, we will still be interested in  $r = 1$ , but with a different set  $\mathcal{Y}$ . In particular, HK has a fingerprint function  $T(K, \cdot)$ , and we define the set  $\mathcal{FP}(K, x) = \{y \in \mathcal{U} \mid T(K, y) \neq T(K, x)\}$ . We will typically write  $\text{fp}_x$  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. [16], 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  $y, z \in \vec{S} \parallel x$  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  $x$

<sup>1</sup>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.

(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 3.4.3), 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 3.5), 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  $R(K, \cdot)$  needed to find elements covering the target’s counter in the  $i^{\text{th}}$  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  $y_1, \dots, y_r$  with  $R(K, y_j)[i] = p_i$ . This holds for any  $i \in [k]$ , and all  $L_i^r$  are independent. Thus, letting  $L^r = \max\{L_1^r, L_2^r, \dots, L_k^r\}$ , we have

$$\Pr[L^r \leq z] = \prod_{i=1}^k \Pr[L_i^r \leq z] = \left( p^r \sum_{t=0}^{z-r} \binom{t+r-1}{t} (1-p)^t \right)^k. \quad (3-1)$$

Note that relation (3-1) fully defines  $z$  for any fixed values of  $\Pr[L^r \leq z]$ ,  $m, k, r$ . Thus, we will be able to relate  $\Pr[\text{Cover}_x^r]$  and  $\Pr[L^r = z]$  via the resources used in attacks, e.g.,  $\text{Cover}_x^r$  occurs iff  $L^r \leq f_{m,k,r}(q_H, q_U, q_Q)$  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

$$\Pr[L^1 \leq z] = \left( (1 - q)(1 + q + q^2 + \dots + q^{z-1}) \right)^k = (1 - q^z)^k \quad (3-2)$$

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

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

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.<sup>2</sup> This constant depends only on parameters  $m$  and  $k$ .

### 3.4.2 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  $\text{Cover}_x^r$  be the event that in the execution of  $\text{Atk}_{\Pi}^{\text{err-fe}[u,v]}(\mathcal{A})$ , the adversary queries the **Up**-oracle with  $\text{up}_{e_t}, \dots, \text{up}_{e_t}$  and  $e_1, \dots, e_t$  is an  $r$ -cover for the target. For concision, define random variable

$\text{Err} = \text{Atk}_{\Pi}^{\text{err-fe}[u,v]}(\mathcal{A})$ . We will mainly focus on  $\mathbb{E}[\text{Err}]$  when analyzing the behavior of structures,

---

<sup>2</sup>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  $x$ .

so here we observe that the non-negative nature of  $\text{Err}$  allows us to write

$\mathbb{E}[\text{Err}] = \sum_{\xi > 1} \Pr[\text{Err} \geq \xi]$ . In determining the needed probabilities, it will be beneficial to condition on  $\text{Cover}_x^r$ , 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,  $\mathbb{E}[\text{Err}]$  may depend on the value of  $Q$ , and then we calculate the expectation as  $\mathbb{E}[\mathbb{E}[\text{Err} | Q]]$ . After a cover  $C$  is found by the first stage (so  $\text{Cover}_x^1$  holds), the second stage can insert  $C$  until the resource budget is exhausted. Note that each insertion of  $C$  will increase the CMS estimation-error by one. Our attacks ensure that  $|C| \leq k$ , and so the number of  $C$ -insertions in the second stage is at least  $\left\lfloor \frac{q_U - Q}{k} \right\rfloor$ . This implies that

$\mathbb{E}[\text{Err} | Q] \geq \sum_{\xi=1}^{\lfloor (q_U - Q)/k \rfloor} \Pr[\text{Err} \geq \xi | Q, \text{Cover}_x^1] \Pr[\text{Cover}_x^1 | Q]$  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

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

**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;  $\mathbb{E}[\text{Err}] = \left\lfloor \frac{q_U}{k} \right\rfloor \Pr[\text{Cover}_x^1 | Q=0]$ . 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 (3-2) we arrive at

$$\Pr [\text{Cover}_x^1 \mid Q=0] = \left(1 - (1 - 1/m)^{\frac{q_H}{k} - 1}\right)^k \quad (3-4)$$

implying  $\mathbb{E}[\text{Err}] \geq \lfloor \frac{q_U}{k} \rfloor \left(1 - (1 - 1/m)^{\frac{q_H}{k} - 1}\right)^k$ . For  $q_H/k \gg 1$ , which is likely as  $q_H$  is *offline* work and practical  $k$  are small,  $\mathbb{E}[\text{Err}] \approx q_U/k$ . The full attack can be found in Figure 3-4.

CoverAttack <sup>Hash, Up, Qry</sup> ( $x, K, \text{repr}$ )	FindCover <sup>Hash</sup> ( $r, x, K$ )
1 : cover $\leftarrow$ FindCover <sup>Hash</sup> (1, $x, K$ )	1 : cover $\leftarrow \emptyset$ ; found $\leftarrow$ False
2 : <b>until</b> $q_U$ <b>Up</b> -queries made:	2 : $\mathcal{I} \leftarrow \emptyset$ ; tracker $\leftarrow$ zeros( $k$ )
3 : <b>for</b> $e \in$ cover: <b>Up</b> ( $e$ )	3 : $\nmid R(K, x)[i] = \text{Hash}(\langle i, K, x \rangle)$
4 : <b>return</b> done	4 : $(p_1, p_2, \dots, p_k) \leftarrow R(K, x)$
	5 : <b>while</b> not found
	6 : <b>if</b> $q_H$ <b>Hash</b> -queries made
	7 : <b>return</b> $\emptyset$
	8 : $y \leftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})$
	9 : $\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}$
	10 : $(q_1, q_2, \dots, q_k) \leftarrow R(K, y)$
	11 : <b>for</b> $i \in [k]$
	12 : <b>if</b> $p_i = q_i$ <b>and</b> tracker[ $i$ ] < $r$
	13 :             cover $\leftarrow$ cover $\cup \{y\}$
	14 :             tracker[ $i$ ] + = 1
	15 : <b>if</b> sum(tracker) = $rk$
	16 :         found $\leftarrow$ True
	17 : <b>return</b> cover

Figure 3-4. Cover Set Attack for the CMS in public hash function setting. We use  $R(K, x)$  to mean  $(\text{Hash}(\langle 1, K, x \rangle), \text{Hash}(\langle 2, K, x \rangle), \dots, \text{Hash}(\langle k, K, x \rangle))$ . The attack is parametrized with the update and **Hash** query budget  $q_U$  and  $q_H$ .

**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.

CoverAttack <sup>Up,Qry</sup> ( $x, \perp, \perp$ )	FindCover <sup>Up,Qry</sup> ( $x$ )
<pre> 1 : cover <math>\leftarrow</math> FindCover<sup>Up,Qry</sup>(<math>x</math>) 2 : <b>until</b> <math>q_U</math> Up-queries made: 3 :   <b>for</b> <math>e \in</math> cover: Up(<math>e</math>) 4 : <b>return</b> done </pre>	<pre> 1 : / find 1-cover for x 2 : cover <math>\leftarrow \emptyset</math> 3 : found <math>\leftarrow</math> False 4 : <math>\vec{I} \leftarrow \emptyset</math>; <math>a \leftarrow</math> Qry(<math>x</math>) 5 : <b>while</b> not found 6 :   <b>if</b> <math>q_U</math> Up- or <math>q_Q</math> Qry-queries made 7 :     <b>return</b> cover 8 :   <math>y \leftarrow \mathcal{U} \setminus (\vec{I} \cup \{x\})</math> 9 :   <math>\vec{I} \leftarrow \vec{I} \cup \{y\}</math> 10 :  Up(<math>y</math>); <math>a' \leftarrow</math> Qry(<math>x</math>) 11 :  <b>if</b> <math>a' \neq a</math> : 12 :    cover <math>\leftarrow \{y\}</math> 13 :    found <math>\leftarrow</math> True 14 :  <b>for</b> <math>i \in [2, 3, \dots, k]</math> <i>lg : CMS : CK : change</i> 15 :    <math>a \leftarrow</math> MinUncover<sup>Up,Qry</sup>(<math>x, a',</math> cover) 16 :    <b>if</b> <math>a =</math> cover : <b>return</b> cover 17 :    <b>for</b> <math>y \in \mathcal{I}</math> / in order of insertion to <math>\mathcal{I}</math> 18 :      <b>if</b> <math>q_U</math> Up- or <math>q_Q</math> Qry-queries made 19 :        <b>return</b> cover 20 :      Up(<math>y</math>); <math>a' \leftarrow</math> Qry(<math>x</math>) 21 :      <b>if</b> <math>a' \neq a</math> : 22 :        cover <math>\leftarrow</math> cover <math>\cup \{y\}</math> 23 :        <math>\vec{I} \leftarrow \mathcal{I} \setminus \{y\}</math> 24 :        <b>break</b> 25 :  <b>return</b> cover </pre>
<pre> MinUncover<sup>Up,Qry</sup>(<math>x, a',</math> cover) 1 : <math>b' \leftarrow -1</math> 2 : <b>while</b> <math>a' \neq b'</math> 3 :   <b>if</b> (<math>q_U -  \text{cover}  + 1</math>)Up- 4 :     or <math>q_Q</math> Qry-queries made: 5 :     <b>return</b> cover 6 :   <math>b' \leftarrow a'</math> 7 :   <b>for</b> <math>y \in</math> cover : Up(<math>y</math>) 8 :   <math>a' \leftarrow</math> Qry(<math>x</math>) 9 : <b>return</b> <math>a'</math> </pre>	

Figure 3-5. 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  $q_Q$ .

In Figure 3-5 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  $(p_1, p_2, \dots, p_k) \leftarrow R(K, x)$  and let

$M[1][p_1] = c_1, \dots, M[k][p_k] = c_k$  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  $C_1 \leftarrow \{z_1\}$ . Notice that by definition, the insertion of  $z_1$  increases the estimation error by one.

Let  $\mathcal{M}(C_1) = \{i \in [k] \mid \exists z \in C : R(K, z)[i] = p_i\}$ , i.e., the set of rows whose  $x$ -counters are covered by  $C_1$ , and let  $\delta_1 = \min_{j \notin \mathcal{M}(C_1)} \{M[j][p_j]\} - \min_{i \in \mathcal{M}(C_1)} \{M[i][p_i]\}$ . Notice that  $\delta_1$  is the gap between the smallest counter(s) *not* covered by  $C_1$ , and the smallest counter(s) that are covered by  $C_1$ . (Observe that  $z_1$  may also cover non-minimal  $x$ -counters.) Thus, if we now reinsert  $C_1$  a total of  $\delta_1$  times, this gap shrinks to zero; reinserting it once more will cause some  $x$ -counter that is *not* covered by  $C_1$  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  $C_1 = \{z_1\}$ , and compute  $\delta_1 = 3 - 1 = 2$ . If we were to insert  $C_1$  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  $C_1$ , 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. ◦

Notice that the  $\delta_1 + 1$  re-insertions of  $C_1$  will increase the CMS estimate of  $n_x$  by exactly  $\delta_1$ . At this point we begin round 2, searching for  $z_2 \in \vec{I} \setminus C_1$  that covers the newly minimal  $x$ -counters. Recall that the elements of  $\vec{I}$  are distinct (by design), so if we reinsert  $\vec{I} \setminus C_1$  *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  $C_2 \leftarrow C_1 \cup \{z_2\}$ , and then switch to reinserting  $C_2$  a total of  $\delta_2 + 1$  times (where  $\delta_2$  is defined analogously to  $\delta_1$ ) to end round 2. Again, this increases the estimation error by  $\delta_2$ .

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  $C_{i-1}$ , and there are only  $k$  counters to cover. Notice that in round  $\ell$ , when we reinsert  $C_\ell$  we will never observe that some new  $x$ -counter has become minimal: all  $x$ -counters are covered by  $C_\ell$ , so all will be increased by each reinsertion. Nonetheless, each reinsertion of  $C_\ell$  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  $C_\ell$  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  $C_\ell$  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

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

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

We would like to develop an expression for  $\mathbb{E}[\text{Err}]$ , 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  $\text{Err}$  with  $\widehat{\text{Err}}$  by replacing  $\ell$  with  $k$ , dropping the flooring operation, arriving at

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

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

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

We do not have a crisp way to describe the distribution of the  $\delta_i$  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); moreover, 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  $C_1 = \{z_1\}$  a total of  $\delta_1 + 1$  times, we switch to reinserting  $\vec{I} \setminus C_1$  until we find a  $z_2$  that covers the currently minimal  $x$ -counters. When we begin to reinsert  $C_2$ , we know by construction that  $\delta_2 \leq \min_{j \notin \mathcal{M}(\{z_1, z_2\})} \{M[j][p_j]\} - (\min_{j \notin \mathcal{M}(\{z_1\})} \{M[j][p_j]\} + 1)$ . 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)  $\alpha$  to that maximum counter value, and the repeated insertions of  $C_1$  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  $\alpha$  is in the neighborhood of  $H_k < 4$ . Continuing this way, we reach the conclusion that the dominant term in  $\mathbb{E}[\widehat{\text{Err}}] \approx \mathbb{E}[\text{Err}]$  will be  $\frac{q_U}{k} - mH_k$ . This is observed experimentally in Table 3-2. 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  $|C_k| = k$ , and we know that a complete cover has been found. Thus, we do not need to make any **Qry**-queries during reinsertions of  $C_k$ . If  $\ell < k$ , however, then we must make **Qry**-queries during reinsertions of  $C_\ell$ , because we do not know that  $C_\ell$  contains a complete cover.

Either way, the number of **Qry**-queries need to reach  $C_\ell$  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  $\delta_i$ , we can safely bound  $\mathbb{E} [\max_{i \in [\ell-1]} \{\delta_i\}]$  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  $C_\ell$  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  $C_\ell$ , 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  $\delta_i$ , it seems likely that if some  $C_i$  is taking a large number of insertions, one can likely assume that  $C_i$  is a complete cover, cease making estimation queries and switch to an insertion only strategy.

**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 3-6). 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  $C$  is found. By the definition of  $L^1$ , the cover is found with  $(q'_U = 1 + L^1)$  **Up**-queries, and is after reinserted  $\lfloor (q_U - q'_U)/|C| \rfloor$  times, each time adding one to the estimation

error. Hence,  $\text{Err} \geq \lfloor (q_U - 1 - L^1)/|C| \rfloor \geq \lfloor (q_U - 1 - L^1)/k \rfloor$  and

$$\mathbb{E}[\text{Err}] \geq \frac{q_U - 1 - \mathbb{E}[L^1]}{k} \approx \frac{q_U - mH_k}{k}.$$

CoverAttack <sup>Up</sup> ( $x, \perp, \text{repr}$ )	FindCover <sup>Up</sup> ( $r, x, \text{repr}$ )
<pre> 1 : cover <math>\leftarrow</math> FindCover<sup>Up</sup>(1, <math>x</math>, repr) 2 : <b>until</b> <math>q_U</math> Up-queries made: 3 :   <b>for</b> <math>e \in \text{cover}</math>: Up(<math>e</math>) 4 : <b>return</b> done </pre>	<pre> 1 : cover <math>\leftarrow \emptyset</math>; found <math>\leftarrow</math> False 2 : <math>\mathcal{I} \leftarrow \emptyset</math>; tracker <math>\leftarrow \text{zeros}(k)</math> 3 : repr' <math>\leftarrow</math> Up(<math>x</math>) 4 : / compute <math>x</math>'s indices 5 : <b>for</b> <math>i \in [k]</math> 6 :   <b>for</b> <math>j \in [m]</math> 7 :     <b>if</b> repr'[<math>i</math>][<math>j</math>] <math>\neq</math> repr[<math>i</math>][<math>j</math>] 8 :       <math>p_i \leftarrow j</math>; <b>break</b>; 9 : <b>while</b> not found 10 :   <b>if</b> <math>q_U</math> Up-queries made : <b>return</b> <math>\emptyset</math> 11 :   <math>y \leftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})</math> 12 :   <math>\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}</math> 13 :   repr <math>\leftarrow</math> repr' 14 :   repr' <math>\leftarrow</math> Up(<math>y</math>) 15 :   / compute <math>y</math>'s indices 16 :   <b>for</b> <math>i \in [k]</math> 17 :     <b>for</b> <math>j \in [m]</math> 18 :       <b>if</b> repr'[<math>i</math>][<math>j</math>] <math>\neq</math> repr[<math>i</math>][<math>j</math>] 19 :         <math>q_i \leftarrow j</math>; <b>break</b>; 20 :   <b>for</b> <math>i \in [k]</math> 21 :     / compare <math>x</math>'s and <math>y</math>'s indices row by row 22 :     <b>if</b> <math>p_i = q_i</math> <b>and</b> tracker[<math>i</math>] <math>&lt; r</math> 23 :       cover <math>\leftarrow</math> cover <math>\cup \{y\}</math> 24 :       tracker[<math>i</math>] <math>+= 1</math> 25 :   <b>if</b> sum(tracker) = <math>rk</math> 26 :     found <math>\leftarrow</math> True 27 : <b>return</b> cover </pre>

Figure 3-6. Cover Set Attack for the CMS in private hash function and public representation setting. The attack is parametrized with the update query budget  $q_U$ .

### 3.4.3 Cover-Set Attacks on HK

By examining the HK pseudocode, it is not hard to see that when a stream  $\vec{S}$  satisfying the NFC condition is inserted in the HK structure, over-estimations are not possible; any error in frequency

estimates is due to underestimation. We also note that if  $\vec{S}$  satisfies the NFC condition, then any cover that it contains for  $x \in \mathcal{U}$  must be a  $(\mathcal{FP}_x, x, r)$ -cover. In attacking HK, we will build  $(\mathcal{FP}_x, x, 1)$ -covers; as such, in this section we will often just say “cover” as shorthand.

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

We note that attacks of this nature would be particularly damaging in instances where the underlying application uses HK to identify the most frequent elements in a stream  $\vec{S}$ . With relatively few insertions of the cover set, one would be able to hide many occurrences of  $x$ . DDoS detection systems, for example, rely on compact frequency estimators to identify communication end-points that are subject to an abnormally large number of incoming connections [24]. In this case, the target  $x$  is an end-point identifier (e.g., an IP address and/or TCP port). Being able to hide the fact that the end-point  $x$  is a “heavy hitter” in the stream of incoming flow destinations could result in  $x$  being DDoSed.

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

**Setting the attack parameter  $t$ .** Say our attack’s resource budget is  $(q_H, q_U, q_Q)$ . The HK attacks find a cover  $C = \{z_1, z_2 \dots\}$  and then inserts it  $t$  times. We set the value  $t$  such the

probability  $p$  of decrementing the any of the target's counters with subsequent insertions of  $x$  is sufficiently small. For our experiments we set  $p = 2^{-128}$ .

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

Say  $f(t) = k (q_U)^t d^{\frac{t(t+1)}{2}}$ . If the attack set  $A[i][p_i].\text{cnt} = t$  with  $A[i][p_i].\text{fp} = \text{fp}_{z_i}, z_i \neq x$  for each  $i$ , then the probability of  $x$  overtaking any of its counters by the end of the attack is bounded by  $\Pr[\bigvee_{i=1}^k D_i^t] \leq f(t)$ .

**Public hash and public representation setting.** This attack (Figure 3-7) is similar to the CMS attack for the public hash setting, but with a few tweaks. The cover is inserted only  $t$  times and then the **Up** budget is exhausted by inserting target  $x$  (at least  $(q_U - tk)$  times) to accumulate error. If  $\neg D^t$  then this process introduces the error of at least  $(q_U - tk)$ . Thus, as the cover finding step uses **Hash** only and induces no error,

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

For the term  $\Pr[\text{Cover}_x^1 \mid Q = 0]$  we can simply apply the same bound as for the CMS attack (Equation (3-4)) obtaining

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

**Private hash and private representation setting.** We present the attack for this setting in Figure 3-8. The attack starts by inserting  $x$  once. Starting with an empty HK implies that then  $x$  owns all of its buckets, i.e.,  $A[i][p_i].\text{fp} = \text{fp}_x$  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

CoverAttack <sup>Hash,Up.Qry</sup> ( $x, K, \text{repr}$ )	FindCover <sup>Hash</sup> ( $x, K$ )
<pre> 1 : cover <math>\leftarrow</math> FindCover<sup>Hash</sup>(<math>x, K</math>) 2 : <math>t \leftarrow \text{Get-t}( \text{cover} )</math> 3 : <b>for</b> <math>e \in \text{cover}</math> 4 :   <b>for</b> <math>i \in [t]</math>: <b>Up</b>(<math>e</math>) 5 : <b>until</b> <math>q_U</math> <b>Up</b>-queries made: 6 :   <b>Up</b>(<math>x</math>) 7 : <b>return</b> done </pre>	<pre> 1 : cover <math>\leftarrow \{\}</math>; found <math>\leftarrow</math> False 2 : <math>\mathcal{I} \leftarrow \emptyset</math>; tracker <math>\leftarrow \text{zeros}(k)</math> 3 : <math>\text{for } R(K, x)[i]</math> 4 :   <math>I = \text{Hash}(\langle \text{"ct"}, i, K, x \rangle)</math> 5 :   <math>(p_1, p_2, \dots, p_k) \leftarrow R(K, x)</math> 6 :   <b>while</b> not found 7 :     <b>if</b> <math>q_H</math> <b>Hash</b>-queries made 8 :       <b>return</b> <math>\emptyset</math> 9 :   <math>y \leftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})</math> 10 :   <math>\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}</math> 11 :   <math>(q_1, q_2, \dots, q_k) \leftarrow R(K, y)</math> 12 :   <b>for</b> <math>i \in [k]</math> 13 :     <b>if</b> <math>p_i = q_i</math> 14 :       <math>\text{remove duplicates}</math> 15 :       cover[<math>i</math>] <math>\leftarrow y</math> 16 :       <b>if</b> tracker[<math>i</math>] &lt; 1 17 :         tracker[<math>i</math>] <math>\leftarrow</math> 1 18 :   <b>if</b> sum(tracker) = <math>k</math> 19 :     found <math>\leftarrow</math> True 20 :   <math>\text{return the cover}</math> 21 : <b>return</b> cover.values() </pre>
<pre> Get-t() 1 : <math>g(t) \leftarrow \log_2(k \cdot (q_U)^t d^{t(t+1)/2}) - \log_2(p)</math> 2 : <math>\text{// find the roots of the negative quadratic polynomial } g</math> 3 : <math>t_1, t_2 \leftarrow \text{FindRootsOf}(g) \text{ // } t_1 \leq t_2</math> 4 : <math>\text{// set } t \text{ so } t \geq 1 \text{ and } g(t) &lt; 0</math> 5 : <b>if</b> <math>t_1 &gt; 1</math> <b>or</b> <math>t_2 &lt; 1</math> : <math>t \leftarrow 1</math> 6 : <b>if</b> <math>t_2 &gt; 1</math> : <math>t \leftarrow \lceil t_2 \rceil</math> 7 : <b>if</b> <math>t_2 = 1</math> : <math>t \leftarrow 2</math> 8 : <b>return</b> <math>t</math> </pre>	

Figure 3-7. Cover Set Attack for the HK in public hash function setting. We use  $R(K, x)$  to mean  $(\text{Hash}(\langle \text{"ct"}, 1, K, x \rangle), \text{Hash}(\langle \text{"ct"}, 2, K, x \rangle), \dots, \text{Hash}(\langle \text{"ct"}, k, K, x \rangle))$ . The attack is parametrized with the update and **Hash** query budget  $q_U$  and  $q_H$ .

keeps inserting *distinct* elements until the frequency estimate for  $x$  drops to 0, i.e.,

$$A[i][p_i].\text{fp} \neq \text{fp}_x \text{ for all rows } i.$$

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  $A[i][p_i].\text{fp}$  from  $\text{fp}_x$  most recently). 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.



CoverAttack <sup>Up,Qry</sup> ( $x, \perp, \perp$ )	FindInsertCover <sup>Up,Qry</sup> ( $x$ )
<pre> 1 : FindInsertCover<sup>Up,Qry</sup>(<math>x</math>) 2 : <b>until</b> <math>q_U</math> Up-queries made: 3 :   Up(<math>x</math>) 4 : <b>return</b> done </pre>	<pre> 1 : // insert <math>\leq k</math> elements <math>t</math> times in a row 2 : cover <math>\leftarrow \emptyset</math> 3 : <math>t \leftarrow \text{Get-t}()</math> 4 : <math>I \leftarrow \emptyset</math> 5 : <b>for</b> <math>i \in [1, 2, \dots, k] : k - \text{priv} - \text{priv} - a - y_i</math> 6 :   Reintro<sup>Up,Qry</sup>(<math>x</math>) 7 :   <b>while</b> True 8 :     <b>if</b> <math>q_U</math> Up- or <math>q_Q</math> Qry-queries made 9 :       <b>return</b> 10 :    <math>y \leftarrow \mathcal{U} \setminus (I \cup \{x\})</math> 11 :    <math>I \leftarrow I \cup \{y\}</math> 12 :    Up(<math>y</math>); <math>a \leftarrow \text{Qry}(x)</math> 13 :    <b>if</b> <math>a = 0</math> : 14 :      cover <math>\leftarrow \text{cover} \cup \{y\}</math> 15 :      <b>for</b> <math>j \in [t] : \text{Up}(y)</math> 16 :      <b>break</b> 17 : <b>return</b> </pre>
<pre> Reintro<sup>Up,Qry</sup>(<math>x</math>) </pre> <hr/> <pre> 1 : // reintroduce target <math>x</math> 2 : <b>while</b> True 3 :   <b>if</b> <math>q_U</math> Up- or <math>q_Q</math> Qry-queries made: 4 :     <b>return</b> 5 :   Up(<math>x</math>); <math>a \leftarrow \text{Qry}(x)</math> 6 :   <b>if</b> <math>a &gt; 0</math> : <b>return</b> 7 : <b>endwhile</b> 8 : </pre>	

Figure 3-8. 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  $\text{Get-t}(\cdot)$  from Figure 3-7.

The adaptive portion of our attack proceeds as follows. In each round  $i = 2, \dots$  we first keep reinserting  $x$  until  $\text{HK}(x)$  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  $C_i$ , by inserting *new distinct* elements, until we find a  $z_i$  that drops  $\text{HK}(x)$  to 0. We set  $C_i \leftarrow C_{i-1} \cup \{z_i\}$  and insert  $z_i$   $t$  times. At this point, all counters  $c_1, \dots, c_k$  are owned by elements  $\neq x$  again, and all are of value one, but the ones covered by  $C_i$  which (very likely) hold a value strictly greater than 1 and (very) close to  $t$ .

The procedure ensures that after some  $\ell \leq k$  rounds we have found a complete 1-cover with (very) high probability. Each round  $i$  adds maximally one new element to the incomplete cover  $C_{i-1}$ . The added element covers whatever  $x$  is owning at the beginning of the round. Thus, with (very) high

probability, counters owned by  $x$  in the round are not covered by  $C_{i-1}$ . This is because all the counters covered by  $C_{i-1}$  were set to value  $t$  (or a value close to  $t$  with very high probability<sup>3</sup>) at some point, and the selection of  $t$  makes the probability of later overtaking one such counter (very) small. There are only  $k$  counters to cover and so with (very high) probability having only  $k$  rounds suffices to find a cover.

Let  $\mathcal{I}_i$  be the set of inserted elements  $\neq x$  in each round. We get the number of **Up**-queries required to complete  $k$  rounds is

$$q'_U \leq \sum_{i=1}^k (d_i + |\mathcal{I}_i| + t) = \sum_{i=1}^k (d_i + |\mathcal{I}_i|) + tk. \quad (3-5)$$

So,  $x$  can be potentially inserted  $q_U - q'_U$  times, accumulating some additional error<sup>4</sup>. Let us assume that  $q_Q$  is not the limiting resource in the attack. Say  $C$  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  $C_{i-1}$ . If, in addition, we have  $|C| = k$ ,  $k$  different elements set  $k$  different counters of  $x$  (i.e. all of  $x$ 's counters) to  $t$  making them impossible to decrement later. Therefore, after the rounds to reach  $C$  are completed every further insertion of  $x$  ( $q_U - q'_U$  of them) increased the error by 1. Note that  $|C| = k$  implies the attack completed exactly  $k$  rounds and

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

---

<sup>3</sup>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  $\geq 0.9$  – 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.

<sup>4</sup>We say potentially as the **Qry**-query budget might be a limiting factor.

Let  $D_i$  be the set of rows  $j$  with  $A[j][p_j].\text{fp} = \text{fp}_x$  (i.e.  $x$  owning the counter), and let  $c_{i,j}$  be the values of  $A[j][p_j].\text{cnt}$  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  $A[j][p_j].\text{fp} \neq \text{fp}_x$ . 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  $j \in D_i$  – that is 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  $\mathbb{E}[|\mathcal{I}_i|] \leq \mathbb{E}[|\mathcal{I}_1|]$ . So,  $\mathbb{E}[|\mathcal{I}_i|] \leq \frac{m}{d} H_k$ . This implies that

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

We expect  $\Pr[\neg(D^t) \wedge |C| = k] \approx 1$  and  $\mathbb{E}[\text{Err}] \approx q_U - tk - \frac{km}{d} H_k$ . We confirmed this experimentally as seen in Table 3-2.

**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  $x$ , such that  $x$  is guaranteed to own all of its counters. Then, we keep adding *distinct*

CoverAttack <sup>Up, Qry</sup> ( $x, \perp, \text{repr}$ )	FindCover <sup>Up</sup> ( $x, \text{repr}$ )
<pre> 1 : cover <math>\leftarrow</math> FindCover<sup>Up</sup>(<math>x, \text{repr}</math>) 2 : <math>t \leftarrow \text{Get-t}()</math> 3 : <b>for</b> <math>e \in \text{cover}</math> 4 :   <b>for</b> <math>i \in [t]</math>: <b>Up</b>(<math>e</math>) 5 : <b>until</b> <math>q_U</math> <b>Up</b>-queries made: 6 :   <b>Up</b>(<math>x</math>) 7 : <b>return</b> done </pre>	<pre> 1 : cover <math>\leftarrow \{\}</math>; found <math>\leftarrow \text{False}</math> 2 : <math>\mathcal{I} \leftarrow \emptyset</math>; tracker <math>\leftarrow \text{zeros}(k)</math> 3 : <math>\text{repr}' \leftarrow \text{Up}(x)</math> 4 : <i>/ compute <math>x</math>'s indices</i> 5 : <b>for</b> <math>i \in [k]</math> 6 :   <b>for</b> <math>j \in [m]</math> 7 :     <b>if</b> <math>\text{repr}'[i][j].\text{fp} \neq \text{repr}[i][j].\text{fp}</math> 8 :       <math>p_i \leftarrow j</math>; <b>break</b>; 9 : <b>while</b> not found 10 : <b>if</b> <math>q_U</math> <b>Up</b>-queries made : <b>return</b> <math>\emptyset</math> 11 : <math>y \leftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})</math> 12 : <math>\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}</math> 13 : <math>\text{repr} \leftarrow \text{repr}'</math> 14 : <math>\text{repr}' \leftarrow \text{Up}(y)</math> 15 : <i>/ compute <math>y</math>'s indices</i> 16 : <b>for</b> <math>i \in [k]</math> 17 :   <math>q_i \leftarrow \text{False}</math> 18 :   <b>for</b> <math>j \in [m]</math> 19 :     <b>if</b> <math>\text{repr}'[i][j].\text{fp} \neq \text{repr}[i][j].\text{fp}</math> 20 :       <math>q_i \leftarrow j</math>; <b>break</b>; 21 : <b>for</b> <math>i \in [k]</math> 22 :   <i>/ compare <math>x</math>'s and <math>y</math>'s indices row by row</i> 23 :   <b>if</b> <math>q_i \neq \text{False}</math> <b>and</b> <math>p_i = q_i</math> 24 :     <i>/ remove duplicates</i> 25 :     <math>\text{cover}[i] \leftarrow y</math> 26 :     <b>if</b> <math>\text{tracker}[i] &lt; 1</math> 27 :       <math>\text{tracker}[i] \leftarrow 1</math> 28 :   <b>if</b> <math>\text{sum}(\text{tracker}) = k</math> 29 :     found <math>\leftarrow \text{True}</math> 30 :   <i>/ cover elements own the target's counters</i> 31 : <b>return</b> <math>\text{cover.values}()</math> <i>/ all counters set to 1</i> </pre>

Figure 3-9. 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  $\text{Get-t}()$  from Figure 3-7.

elements, until all the  $A[i][p_i].fp$  that once belonged to  $x$  has changed, in turn signaling the cover for  $x$  has been found. We give a pseudocode description of this attack in Figure 3-9.

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

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

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

### 3.5 Count-Keeper

In Figure 3-10 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.

### 3.5.1 Structure

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

To insert a stream element  $x$  arrives, we run the CMS and HK update procedures

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

When queried for the frequency estimate of an element  $x \in \mathcal{U}$ , CK first computes the CMS and HK estimates, which we will write as  $\text{CMS}(x)$  and  $\text{HK}(x)$  for brevity. If  $\text{CMS}(x) = \text{HK}(x)$ , 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)  $\text{HK}(x) \leq n_x \leq \text{CMS}(x)$ . 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  $A[i][p_i].\text{fp}$  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,  $\text{CK}(x)$  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

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

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

Figure 3-10. Keyed structure  $\text{CK}[R, T, m, k]$  supporting point-queries for any potential stream element  $x$  ( $\text{qry}_x$ ).  $\text{QRY}_K^{\text{CMS}}$ ,  $\text{UP}_K^{\text{CMS}}$ , resp.  $\text{QRY}_K^{\text{HK}}$ ,  $\text{UP}_K^{\text{HK}}$ , denote query and update algorithms of keyed structure  $\text{CMS}[R, T, m, k]$  (Figure 3-2), resp.  $\text{HK}[R, T, m, k, 1]$  (Figure 3-3, but note  $d = 1$ ). The parameters are a function  $R : \mathcal{K} \times \{0, 1\}^* \rightarrow [m]^k$ , a function  $T : \mathcal{K} \times \{0, 1\}^* \rightarrow \{0, 1\}^n$  for some desired fingerprint length  $n$ , and integers  $m, k \geq 0$ . A concrete scheme is given by a particular choice of parameters.

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  $\Theta_1^i$ , and when  $i \in I_x$  the estimator is  $\Theta_2^i$ . 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  $\text{CMS}(x)$ .

### 3.5.2 Correcting CMS and Correctness of CK

In what follows, we will assume the NFC condition. For sufficiently large fingerprints (e.g.,  $\tau$ -bit fingerprints where  $2^\tau$  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  $n_x \leq \sum_{y \in V_x^i} n_y$ . 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  $A[i][p_i].\text{cnt}$ . So if  $n_x > \sum_{y \in V_x^i} n_y$ , the fingerprint held at  $A[i][p_i]$  would be  $\text{fp}_x$ . 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,  $n_x + A[i][p_i] - 1 \leq \sum_{y \in V_x^i} n_y$ , which implies that

$$M[i][p_i] = n_x + \sum_{y \in V_x^i} n_y \leq 2n_x + A[i][p_i] - 1.$$

**Lemma 3-1.** *Let  $\vec{S}$  satisfy the NFC condition, and let  $x \in \mathcal{U}$ . Then for any  $i \in \hat{I}_x$  we have*

$$n_x \leq \frac{M[i][p_i] - A[i][p_i].\text{cnt} + 1}{2} = \Theta_1^i. \quad \blacklozenge$$

*Proof of Lemma 3-1.* We can think of the counter  $A[i][p_i].\text{cnt}$  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  $A[i][p_i].\text{cnt} = 0$  then the stack is empty; then push an  $\text{fp}_x$ -labeled plate and set

$$A[i][p_i].\text{cnt} \leftarrow 1, A[i][p_i].\text{fp} \leftarrow \text{fp}_x.$$



- 2(a). if  $A[i][p_i].\text{cnt} = c > 0$  and  $A[i][p_i].\text{fp} = \text{fp}_x$ , then push an  $\text{fp}_x$ -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 ( $\text{fp}$ -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  $\text{fp}_x$ -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  $\text{fp}_z$ -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  $\text{fp}_z$ -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  $\text{fp}_x$ -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  $\text{fp}_x$ . 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.  $\square$

As this lemma holds for every  $i \in \hat{I}_x$ , we conclude that

$$n_x \leq \Theta_1 = \min_{i \in \hat{I}_x} \{\Theta_1^i\} \leq \min_{i \in \hat{I}_x} \{M[i][p_i]\}.$$

**Correcting  $M[i][p_i]$  when  $x$  does “own”  $A[i][p_i]$ .** Now, say that row  $i \in I_x$ . Under the NFC condition  $A[i][p_i].\text{cnt}$  stores the number of occurrences of  $x$  that are *not* canceled out by occurrences of  $y \in V_x^i$ . So, we must have had at least  $\sum_{y \in V_x^i} n_y \geq n_x - A[i][p_i].\text{cnt}$  occurrences of  $y \in V_x^i$ . This implies  $M[i][p_i] \geq 2n_x - A[i][p_i].\text{cnt}$ , and, by rearranging,

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

**Lemma 3-2.** *Let  $\vec{S}$  satisfy the NFC condition, and let  $x \in \mathcal{U}$ . Then for any  $i \in I_x$  we have*

$$n_x \leq \frac{M[i][p_i] + A[i][p_i].\text{cnt}}{2} = \Theta_2^i. \quad \blacklozenge$$

*Proof of Lemma 3-2.* 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 3-1. View an insertion of  $x$  being associated with either an insertion of  $y \in V_x^i$  that pops off its  $\text{fp}_x$ -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

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

□

As this lemma holds for every  $i \in I_x$ , we conclude that

$n_x \leq \Theta_2 = \min_{i \in I_x} \{\Theta_2^i\} \leq \min_{i \in I_x} \{M[i][p_i]\}$ . Combined with the conclusion of Lemma 3-1, we have  $n_x \leq \text{CK}(x) = \lfloor \min\{\Theta_1, \Theta_2\} \rfloor \leq \text{CMS}(x)$ .

**Precise estimation when some  $|V_x^i| \in \{0, 1\}$ .** If there exists an  $i$  such that  $|V_x^i| = 0$ , then

$M[i][p_i] = A[i][p_i] = n_x$ . Hence, in this special case, both  $\text{CMS}(x) = n_x$  and  $\text{HK}(x) = n_x$ . When this is not the case,  $n_x < M[i][p_i]$  for all  $i \in [k]$ , so  $n_x < \text{CMS}(x)$ . For CK, on the other hand, if there exists a row  $i$  such that  $|V_x^i| = 1$ , we still have  $\text{CK}(x) = n_x$ . Our next result, which is a corollary of Lemmas 3-1 and 3-2, shows that either one of  $\Theta_1^i$  or  $\Theta_2^i$  is precisely  $n_x$ , or the smaller of the two is  $n_x \pm 1/2$ . Thus  $\text{CK}(x) = \lfloor \min\{\Theta_1, \Theta_2\} \rfloor = n_x$ .

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

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

*Proof of corollary 1.* We 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 3-1 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  $i \in \hat{I}_x$  ( $A[i][p_i].\text{fp} \neq \text{fp}_z$ ). Say  $x$  at some point owned the counter. Then, the plate at the bottom of the stack (labeled with  $\text{fp}_z$ ) corresponds to a occurrence of  $z$  that popped off a plate of  $x$ . So, only  $A[i][p_i].\text{cnt} - 1$  occurrences of  $z$  are not associated with  $x$  implying  $n_z = n_x + A[i][p_i].\text{cnt} - 1$ . Hence,  $M[i][p_i] - n_x = n_x + A[i][p_i].\text{cnt} - 1$ , or equivalently,  $n_x = \frac{M[i][p_i] - A[i][p_i].\text{cnt} + 1}{2}$ . 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  $n_z = n_x + A[i][p_i].\text{cnt}$ , and  $n_x = \frac{M[i][p_i] - A[i][p_i].\text{cnt}}{2}$ .

Let now  $i \in I_x$  ( $A[i][p_i].\text{fp} = \text{fp}_x$ ). 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,

$n_x = n_z + A[i][p_i].\text{cnt}$  and, adding  $n_x$  to both sides and rearranging,  $n_x = \frac{M[i][p_i] + A[i][p_i].\text{cnt}}{2}$ . Say  $z$  at some point owned the counter. Then, the plate at the bottom of the stack (labeled with  $\text{fp}_x$ ) corresponds to the occurrence of  $x$  that popped off a plate of  $z$ , and  $n_x = n_z + A[i][p_i].\text{cnt} - 1$  and  $n_x = \frac{M[i][p_i] + A[i][p_i].\text{cnt} - 1}{2}$ . □

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

### 3.5.3 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  $\Theta_1$  and the plain CMS estimate.

**Theorem 3-1.** Fix an  $x \in \mathcal{U}$ , and let  $i^*$  be any row index such that  $\text{CMS}(x) = M[i^*][p_{i^*}]$ . If  $i^* \in \hat{I}_x$  then either  $\text{CK}(x) = n_x$ , or  $\left(\Theta_1 \leq \frac{\text{CMS}(x)}{2}\right)$ .  $\blacklozenge$

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

Next, a similar theorem relating  $\Theta_2$ , the plain CMS estimate, and the HK estimate (when  $d = 1$ ).

**Theorem 3-2.** Fix an  $x \in \mathcal{U}$ , and let  $i^*$  be any row index such that  $\text{CMS}(x) = M[i^*][p_{i^*}]$ . If  $i^* \in I_x$  then either  $\text{CK}(x) = n_x$  or  $\left(\Theta_2 \leq \frac{\text{CMS}(x) + \text{HK}(x)}{2}\right)$ .  $\blacklozenge$

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

$$\Theta_2 = \min_{i \in I_x} \Theta_2^i \leq \Theta_2^{i^*} = \frac{M[i^*][p_{i^*}] + A[i^*][p_{i^*}].\text{cnt}}{2} \leq \frac{\text{CMS}(x) + \text{HK}(x)}{2}. \quad \square$$

Now, if  $\text{CK}(x)$  is determined by line 10 of Figure 3-10, then  $\text{CK}(x) = \frac{\text{CMS}(x) + \text{HK}(x)}{2}$ . On the other hand, if  $\text{CK}(x)$  is determined by line 15, then  $\text{CK}(x) = 0 \leq \frac{\text{CMS}(x) + \text{HK}(x)}{2}$ . If neither of these holds,  $\text{CK}(x) = \lfloor \min\{\Theta_1, \Theta_2\} \rfloor$ . Thus, Theorem 3-1 and 3-2 imply

$$\lfloor \min\{\Theta_1, \Theta_2\} \rfloor \leq \frac{\text{CMS}(x) + \text{HK}(x)}{2}, \text{ giving us the following lemma.}$$

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

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 satisfies the NFC condition, then

$$\text{CK}(x) - n_x \leq \frac{\text{CMS}(x) - \text{HK}(x)}{2}. \quad \blacklozenge$$

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

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

□

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

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

This last observation surfaces something that CK can provide, and neither CMS nor HK can: the ability to signal when the incoming stream is atypical. We explore this in detail in Section 3.5.6.

### 3.5.4 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 [16, 25, 26, 27, 15, 28, 29]. 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<sup>5</sup>. 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) [30]. 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 [31]. It is of interest to

---

<sup>5</sup><http://fimi.uantwerpen.be/data/>

measure compact frequency estimators performance against data following a Zipf distribution [28, 32, 16, 26, 27, 29]. 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.

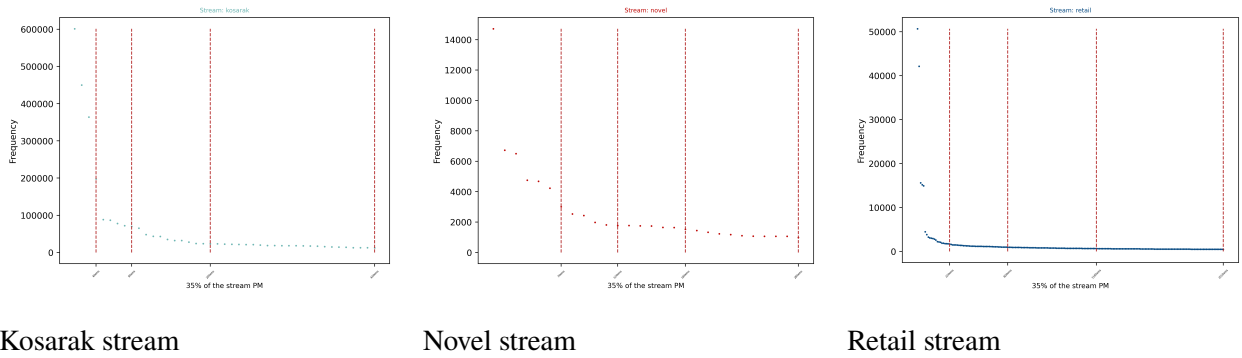


Figure 3-11. We plot the top 35% probability mass for each stream. That is the most frequent elements that make up 35% of the total weight of the stream (i.e. the fewest number of elements in each stream whose frequencies sum to such that when divided by the total length of the stream equal 35%). The 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  $\vec{S} \subseteq \{e_1, e_2, \dots, e_M\}$  with associated frequencies  $(n_{e_1}, n_{e_2}, \dots, n_{e_M})$  we can



order the elements  $\{e_1^*, e_2^*, \dots, e_M^*\}$  such that  $(n_{e_1}^* \geq n_{e_2}^* \geq \dots \geq n_{e_M}^*)$ . Then for some  $K \in \mathbb{Z}^+$  we output the set of elements  $\{e_1^*, e_2^*, \dots, e_K^*\}$  with the  $K$  highest frequencies  $(n_{e_1}^* \geq n_{e_2}^* \geq \dots \geq n_{e_K}^*)$ .

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 [33] 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 [16, 25, 27].

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 [16] 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 [34]. 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 [28].

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 3-11. 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:  $\text{SIS} = |\mathcal{K} \cap \tilde{\mathcal{K}}|$ . This is a 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 [35]. We use the statistic to determine the similarity of the true top- $K$  set  $\mathcal{K}$  of the stream and the estimated top- $K$  set  $\tilde{\mathcal{K}}$  as reported by the CFE. It is defined as  $\text{JI} = \frac{|\mathcal{K} \cap \tilde{\mathcal{K}}|}{|\mathcal{K} \cup \tilde{\mathcal{K}}|}$ . A JI can be in the range  $[0, 1]$ , with a JI of 1 implying a perfect characterization of the true top- $K$  set by the CFE in its top- $K$  estimation.
3. **Minimal Top- $\tilde{K}$  to Capture True Top- $K$  (MCT):** This measures determines the minimal size  $L \geq K$  the estimated top- $k$  set  $\tilde{\mathcal{K}}$  would need to be to capture all elements contained in the true top- $K$  set  $\mathcal{K}$ . That is if one were to order the frequency estimates of all items made by a particular CFE, we would determine the number of items one would need to examine (starting from the most-frequent going down to the least-frequent) until all the elements from  $\mathcal{K}$  were contained in that ordered set. Thus,  $L - K$  indicates the number of elements that fall out of  $\mathcal{K}$  that are incorrectly being individually estimated to be greater than at least one element that is truly in  $\mathcal{K}$ .

4. **Average Relative Error on Top-k elements (ARE):** Average Relative Error is a standard measure to use when comparing CFEs [16]. 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<sup>6</sup>. 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 [14] and satisfies the desired properties of the exponential decay function stated in [16].

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

---

<sup>6</sup>Source code is available at: <https://github.com/smarmy7CD/cfe-in-adv-envs>

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.

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

Table 3-1. 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 3-1. 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 very near 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 3-11. 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 in this particular task. In fact, CK performed better than CMS when not burdened with *very* tiny space constraints. It is able to perfectly estimate the true top- $K$  for all streams over all trials with only 2730 individual counters in the standard parameter setting, while also being adversarial robust where the others structures are not.

### 3.5.5 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.  $\text{CK}(x)$  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  $C$  for  $x$  contains elements  $\{y_1, y_2, \dots, y_t\}$  such that for every counter  $x$  maps to in positions  $(p_1, p_2, \dots, p_k) \leftarrow R(K, x)$  it is such that at least two distinct elements in  $C$  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<sup>7</sup>), then the  $\Theta_1$  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  $\omega$  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 insertions 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  $\Theta_1$  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  $\omega$  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, in addition to the assumption that we have already locked-down the counters of the target with initial insertions of the cover in the structure, we will achieve an error on the target of  $\omega$  – 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 3-12) 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 insertion of the 2-cover increases the error by one. The two cover can be inserted at least  $\frac{qu}{2k}$  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

---

<sup>7</sup>Save for the trivial case in the public representation, private hash setting when no cover is able to be found.

CoverAttack <sup>Hash,Up,Qry</sup> ( $x, K, \text{repr}$ )	FindCover <sup>Hash</sup> ( $r, x, K$ )
<pre> 1 : cover ← FindCover<sup>Hash</sup>(2, <math>x, K</math>) 2 : <b>until</b> <math>q_U</math> <b>Up</b>-queries made: 3 :   <b>for</b> <math>e \in \text{cover}</math>: <b>Up</b>(<math>e</math>) 4 : <b>return</b> done </pre>	<pre> 1 : cover ← <math>\emptyset</math>; found ← False 2 : <math>\mathcal{I} \leftarrow \emptyset</math>; tracker ← zeros(<math>k</math>) 3 : <math>\neg R(K, x)[i] = \text{Hash}(\langle i, K, x \rangle)</math> 4 : <math>(p_1, p_2, \dots, p_k) \leftarrow R(K, x)</math> 5 : <b>while</b> not found 6 :   <b>if</b> <math>q_H</math> <b>Hash</b>-queries made 7 :     <b>return</b> <math>\emptyset</math> 8 :   <math>y \leftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})</math> 9 :   <math>\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}</math> 10 :  <math>(q_1, q_2, \dots, q_k) \leftarrow R(K, y)</math> 11 :  <b>for</b> <math>i \in [k]</math> 12 :    <b>if</b> <math>p_i = q_i</math> <b>and</b> tracker[<math>i</math>] &lt; <math>r</math> 13 :      cover ← cover <math>\cup \{y\}</math> 14 :      tracker[<math>i</math>] + = 1 15 :  <b>if</b> sum(tracker) = <math>rk</math> 16 :    found ← True 17 : <b>return</b> cover </pre>

Figure 3-12. 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  $q_H$ .

cover-finding step, as again, we now find a 2-cover. Assuming  $q_U > 2k$  (so that any found  $C$  can be inserted at least once) we arrive at  $\mathbb{E}[\text{Err}] \geq \lfloor \frac{q_U}{2k} \rfloor \Pr \left[ L^2 \leq \frac{q_H - k}{k} \right]$  Using results from Section 3.4.1 we can further obtain a concrete expression for  $\Pr \left[ L^2 \leq \frac{q_H - k}{k} \right]$ .

### Private hash and representation setting.

Our CK attack for the setting (Figure 3-13) is essentially the same as the CMS attack, except a 2-cover (as opposed to a 1-cover) is detected and repeatedly inserted to build up the error. Using analysis similar to the CMS case and assuming  $q_Q$  is not the limiting factor,

$$\text{Err} \geq \left| \left( \frac{\ell+1}{2} + \frac{1}{\ell} \left( q_U + \sum_{i=1}^{\ell-1} (\ell-i) \delta_i \right) - L^2 \right) \right|$$



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

Figure 3-13. Cover Set Attack for the CK in private hash function and representation setting. The attack is parametrized with the update query and query query budget –  $q_U$  and  $q_Q$ .

with  $\ell \leq 2k$  rounds to find a 2-cover. The error bound is similar to the one for the CMS attack, but with  $L^1$  replaced with  $L^2$  as now  $|\vec{I}|$  is precisely  $L^2$ .

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

expect the following to approximate  $\mathbb{E}[\text{Err}]$ :

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

**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 3-14) 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,

$$\mathbb{E}[\text{Err}] \geq \frac{q_U - 1 - \mathbb{E}[L^2]}{2k} \gtrapprox \frac{q_U - 1 - 2mH_k}{2k}.$$

**Attack Comparisons.** We implemented our attacks against all structures in all settings to

	<b>Public Hash Setting</b>			<b>Private Hash, Private Rep Setting</b>		
<b>Structure</b>	cov	Exp. Err	$\mathbb{E}[\text{Err}]$	cov	Exp. Err	$\mathbb{E}[\text{Err}]$
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 3-2. 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  $\mathbb{E}[\text{Err}]$  according to our analysis.

experimentally verify their correctness and our analysis. In Table 3-2 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

CoverAttack <sup>Up, Qry</sup> ( $x, \perp, \text{repr}$ )	FindCover <sup>Up</sup> ( $r, x, \text{repr}$ )
<pre> 1 : cover <math>\leftarrow</math> FindCover<sup>Up</sup>(2, <math>x</math>, repr) 2 : <b>until</b> <math>q_U</math> Up-queries made: 3 :   <b>for</b> <math>e \in \text{cover}</math>: Up(<math>e</math>) 4 : <b>return</b> done </pre>	<pre> 1 : <math>\langle M, A \rangle \leftarrow \text{repr}</math> 2 : cover <math>\leftarrow \emptyset</math>; found <math>\leftarrow</math> False 3 : <math>\mathcal{I} \leftarrow \emptyset</math>; tracker <math>\leftarrow \text{zeros}(k)</math> 4 : <math>\langle M', A' \rangle \leftarrow \text{Up}(x)</math> 5 : / compute <math>x</math>'s indices 6 : <b>for</b> <math>i \in [k]</math> 7 :   <b>for</b> <math>j \in [m]</math> 8 :     <b>if</b> <math>M'[i][j] \neq M[i][j]</math> 9 :       <math>p_i \leftarrow j</math>; <b>break</b>; 10 : <b>while</b> not found 11 :   <b>if</b> <math>q_U</math> Up-queries made : <b>return</b> <math>\emptyset</math> 12 :   <math>y \leftarrow \mathcal{U} \setminus (\mathcal{I} \cup \{x\})</math> 13 :   <math>\mathcal{I} \leftarrow \mathcal{I} \cup \{y\}</math> 14 :   <math>\langle M, A \rangle \leftarrow \langle M', A' \rangle</math> 15 :   <math>\langle M', A' \rangle \leftarrow \text{Up}(y)</math> 16 :   / compute <math>y</math>'s indices 17 :   <b>for</b> <math>i \in [k]</math> 18 :     <b>for</b> <math>j \in [m]</math> 19 :       <b>if</b> <math>M'[i][j] \neq M[i][j]</math> 20 :         <math>q_i \leftarrow j</math>; <b>break</b>; 21 :   <b>for</b> <math>i \in [k]</math> 22 :     / compare <math>x</math>'s and <math>y</math>'s indices row by row 23 :     <b>if</b> <math>p_i = q_i</math> <b>and</b> tracker[<math>i</math>] &lt; <math>r</math> 24 :       cover <math>\leftarrow</math> cover <math>\cup \{y\}</math> 25 :       tracker[<math>i</math>] += 1 26 :   <b>if</b> sum(tracker) = <math>rk</math> 27 :     found <math>\leftarrow</math> True 28 : <b>return</b> cover </pre>

Figure 3-14. Cover Set Attack for the CK in private hash function and public representation setting. The attack is parametrized with the update query budget  $q_U$ .

sufficient error in each attack to showcase overall trends. We run each attack setting and structure pairing over 100 trials, selecting a random target in each trial, and average the results.

Observe the pattern that when holding  $k$  constant and setting reasonable  $m$  values, adjusting such that CMS, CK, and HK use the same space, attacks against CK generate the least amount of error.

The attacks against CK produce about half of the amount of error as opposed to the CMS attacks, and about  $q_U - \frac{q_U}{2k}$  less the amount of error as opposed to the HK attacks. Moreover, observe that our analytical results closely match those of our experimental results.

### 3.5.6 Adversarial Robustness

Corollary 2 shows that the error in  $\text{CK}(x)$  is largest when  $\text{HK}(x) \ll \text{CMS}(x)$ . In particular, when  $x$  does not own any of its counters  $\text{HK}(x)$  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  $\text{CK}(x)$ .

Specifically, recall that  $\text{CK}(x) = \lfloor \min\{\Theta_1, \Theta_2\} \rfloor$ , where  $\Theta_1$  is the smallest upperbound on  $n_x$  that we can determine by looking only at the rows that  $x$  does not own, and  $\Theta_2$  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  $\text{CK}(x)$ . Dropping the floor for brevity, if  $\text{CK}(x) = \Theta_1$  then Lemma 3-2 tells us that  $\Delta \leq (M[i^*][p_{i^*}] - A[i^*][p_{i^*}].\text{cnt} + 1)/2$ , where  $i^* \in \{j \mid \Theta_1^j = \min_{i \in \hat{I}_x} \{\Theta_1^i\}\}$ .

Likewise, if  $\text{CK}(x) = \Theta_2$  then by Lemma 3-2 we have  $n_x \leq (M[i^*][p_{i^*}] + A[i^*][p_{i^*}].\text{cnt})/2$ , where now  $i^* \in \{j \mid \Theta_2^j = \min_{i \in I_x} \{\Theta_2^i\}\}$ . In this case  $A[i^*][p_{i^*}].\text{cnt} \leq n_x$ , so we know that  $\Delta \leq (M[i^*][p_{i^*}] + A[i^*][p_{i^*}].\text{cnt})/2 - A[i^*][p_{i^*}].\text{cnt} = (M[i^*][p_{i^*}] - A[i^*][p_{i^*}].\text{cnt})/2$ . 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  $\text{QRY}(\text{qry}_x)$  computes  $\Delta$ , 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  $\psi$  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  $\text{CMS}(x)$  is such that  $\Pr[\text{CMS}(x) - n_x \leq \epsilon N] \geq 1 - \delta$  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  $A[i^*][p_{i^*}].\text{cnt}$  will have a value very small relative to the value in  $M[i^*][p_{i^*}]$ . In the worst case  $A[i][p_{i^*}].\text{cnt} = 1$  – in our attacks we force this to be the case. Taking  $A[i^*][p_{i^*}].\text{cnt} \approx 0$ , observe that whether  $\text{CK}(x)$  is determined by  $\Theta_1$  or  $\Theta_2$ , we see  $\text{CK}(x) \approx (1/2)M[i^*][p_{i^*}] \approx (1/2)\text{CMS}(x)$  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  $\epsilon$  from parameter  $m$ ) can be a useful starting point for setting  $\psi$ . As a caveat, however, as  $N$  becomes large, an adversarial stream may be able to induce significant error by setting  $\psi$  in this way (due to the looseness of the CMS bound). Depending on the deployment scenario, smaller values of  $\psi$ , 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 3-15), 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 on the frequency estimation of the target element.

For comparison, we also ran 100 trials, with the same parameters, using the non-adversarial streams from Section 3.5.4. 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.

$\text{REP}_K(S)$	$\text{QRY}_K(\text{repr}, \text{qry}_x)$
<pre> 1 : <math>M \leftarrow \text{zeros}(k, m)</math> 2 : <b>for</b> <math>i \in [k]</math> 3 :   <math>A[i] \leftarrow [(\star, 0)] \times m</math> 4 : <math>\text{repr} \leftarrow \langle M, A \rangle</math> 5 : <b>for</b> <math>x \in S</math> 6 :   <math>\text{repr} \leftarrow \text{UP}_K(\text{repr}, \text{up}_x)</math> 7 : <b>return</b> <math>\text{repr}</math> </pre>	<pre> 1 : <math>\langle M, A \rangle \leftarrow \text{repr}</math> 2 : <math>(p_1, \dots, p_k) \leftarrow R(K, x)</math>, <math>\text{fp}_x \leftarrow T(K, x)</math> 3 : <math>\Theta_1, \Theta_2, \Delta \leftarrow \infty</math> 4 : <math>\text{flag} \leftarrow \text{False}</math> 5 : <math>N \leftarrow \sum_{j=1}^m M[1][j]</math> 6 : <math>\text{cnt}_{\text{UB},x} \leftarrow \text{QRY}_{\text{CMS}_K}(M, \text{qry}_x)</math> 7 : <math>\text{cnt}_{\text{LB},x} \leftarrow \text{QRY}_K^{\text{HK}}(A, \text{qry}_x)</math> 8 : <b>if</b> <math>\text{cnt}_{\text{UB},x} = \text{cnt}_{\text{LB},x}</math> 9 :   <b>return</b> <math>\text{cnt}_{\text{UB},x}, \text{flag}</math> 10 : <b>for</b> <math>i \in [k]</math> 11 :   <b>if</b> <math>A[i][p_i].\text{fp} = \star</math> 12 :     <math>\text{cnt}_{\text{UB},x} \leftarrow 0</math> 13 :     <b>return</b> 0, flag 14 :   <b>else if</b> <math>A[i][p_i].\text{fp} \neq \text{fp}_x</math> 15 :     <math>\Theta \leftarrow \frac{M[i][p_i] - A[i][p_i].\text{cnt} + 1}{2}</math> 16 :     <math>\Theta_1 \leftarrow \min\{\Theta_1, \Theta\}</math> 17 :     <math>\hat{\Delta} \leftarrow \frac{M[i][p_i] - A[i][p_i].\text{cnt} + 1}{2}</math> 18 :     <math>\Delta \leftarrow \min\{\Delta, \hat{\Delta}\}</math> 19 :   <b>else if</b> <math>A[i][p_i].\text{fp} = \text{fp}_x</math> 20 :     <math>\Theta \leftarrow \frac{M[i][p_i] + A[i][p_i].\text{cnt}}{2}</math> 21 :     <math>\Theta_2 \leftarrow \min\{\Theta_2, \Theta\}</math> 22 :     <math>\hat{\Delta} \leftarrow \frac{M[i][p_i] - A[i][p_i].\text{cnt}}{2}</math> 23 :     <math>\Delta \leftarrow \min\{\Delta, \hat{\Delta}\}</math> 24 : <math>\text{cnt}_{\text{UB},x} \leftarrow \lfloor \min\{\Theta_1, \Theta_2\} \rfloor</math> 25 : <b>if</b> <math>\Delta \geq \psi N</math> 26 :   <math>\text{flag} \leftarrow \text{True}</math> 27 : <b>return</b> <math>\text{cnt}_{\text{UB},x}, \text{flag}</math> </pre>
<pre> 1 : <math>\langle M, A \rangle \leftarrow \text{repr}</math> 2 : <math>M \leftarrow \text{UP}_K^{\text{CMS}}(M, \text{up}_x)</math> 3 : <math>A \leftarrow \text{UP}_K^{\text{HK}}(A, \text{up}_x)</math> 4 : <b>return</b> <math>\text{repr} \leftarrow \langle M, A \rangle</math> </pre>	

Figure 3-15. Keyed structure  $\text{CK}[R, T, m, k, \psi]$  supporting point-queries for any potential stream element  $x$  ( $\text{qry}_x$ ) and the ability to raise a flag on “bad” frequency estimation.  $\text{QRY}_K^{\text{CMS}}$ ,  $\text{UP}_K^{\text{CMS}}$ , resp.  $\text{QRY}_K^{\text{HK}}$ ,  $\text{UP}_K^{\text{HK}}$ , denote query and update algorithms of keyed structure  $\text{CMS}[R, T, m, k]$  (Figure 3-2), resp.  $\text{HK}[R, T, m, k, 1]$  (Figure 3-3). The parameters are a function  $R : \mathcal{K} \times \{0, 1\}^* \rightarrow [m]^k$ , a function  $T : \mathcal{K} \times \{0, 1\}^* \rightarrow \{0, 1\}^n$  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.

CHAPTER 4  
PROBABILISTIC DATA STRUCTURES IN THE WILD: A SECURITY ANALYSIS OF REDIS

TODO!

CHAPTER 5  
PROVABLY ROBUST SKIPPING-BASED PROBABILISTIC DATA STRUCTURES

TODO!



## CHAPTER 6 CONCLUSION AND FUTURE WORK

**The following needs to be re-organized and re-written.**

## LIST OF REFERENCES

- [1] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
- [2] B. M. Maggs and R. K. Sitaraman, “Algorithmic nuggets in content delivery,” *ACM SIGCOMM CCR*, vol. 45, p. 52–66, July 2015.
- [3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A distributed storage system for structured data,” in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.
- [4] B. Goodwin, M. Hopcroft, D. Luu, A. Clemmer, M. Curmei, S. Elnikety, and Y. He, “Bitfunnel: Revisiting signatures for search,” in *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017.
- [5] “Bip 37.” <https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki>.
- [6] M. Naor and E. Yogev, “Bloom filters in adversarial environments,” in *Annual Cryptology Conference*, 2015.
- [7] D. Clayton, C. Patton, and T. Shrimpton, “Probabilistic data structures in adversarial environments,” in *ACM SIGSAC CCS*, 2019.
- [8] M. Filić, K. Paterson, A. Unnikrishnan, and F. Virdia, “Adversarial correctness and privacy for probabilistic data structures,” in *ACM SIGSAC CCS*, 2022.
- [9] K. G. Paterson and M. Raynal, “Hyperloglog: Exponentially bad in adversarial settings,” in *EuroS&P*, 2022.
- [10] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, “Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm,” in *DMTCS Conference on Analysis of Algorithms*, 2007.
- [11] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True in *AMC SIGCOMM*, 2000.
- [12] A. Lakhina, M. Crovella, and C. Diot, “Characterization of network-wide anomalies in traffic flows,” in *ACM SIGCOMM Conference on Internet Measurement*, 2004.
- [13] L. Melis, G. Danezis, and E. De Cristofaro, “Efficient private statistics with succinct sketches,” *arXiv preprint arXiv:1508.06110*, 2015.
- [14] “Redisbloom: Probabilistic data structures for redis.” <https://oss.redis.com/redisbloom/>.
- [15] G. Cormode and S. Muthukrishnan, “An improved data stream summary: the count-min sketch and its applications,” *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

- [16] T. Yang, H. Zhang, J. Li, J. Gong, S. Uhlig, S. Chen, and X. Li, “Heavykeeper: An accurate algorithm for finding top- $k$  elephant flows,” *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 1845–1858, 2019.
- [17] B. Sigurleifsson, A. Anbarasu, and K. Kangur, “An overview of count-min sketch and its applications.” <https://easychair.org/publications/preprint/gNlw>, 2019.
- [18] “Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker..” <https://redis.io/>.
- [19] A. Hassidim, H. Kaplan, Y. Mansour, Y. Matias, and U. Stemmer, “Adversarially robust streaming algorithms via differential privacy,” in *NeurIPS*, 2020.
- [20] M. Hardt and D. P. Woodruff, “How robust are linear sketches to adaptive inputs?,” in *ACM STOC*, 2013.
- [21] E. Cohen, X. Lyu, J. Nelson, T. Sarlós, M. Shechner, and U. Stemmer, “On the robustness of counts sketch to adaptive inputs.” <https://arxiv.org/abs/2202.13736>, 2022.
- [22] O. Ben-Eliezer, R. Jayaram, D. P. Woodruff, and E. Yogev, “A framework for adversarially robust streaming algorithms,” *Journal of the ACM*, vol. 69, no. 2, 2022.
- [23] S. A. Markelon, M. Filić, and T. Shrimpton, “Compact frequency estimators in adversarial environments.” Cryptology ePrint Archive, Paper 2023/1366, 2023.
- [24] H. Liu, Y. Sun, and M. S. Kim, “Fine-grained ddos detection scheme based on bidirectional count sketch,” in *International Conference on Computer Communications and Networks*, 2011.
- [25] A. Mandal, H. Jiang, A. Shrivastava, and V. Sarkar, “Topkapi: parallel and fast sketches for finding top- $k$  frequent elements,” *NeurIPS*, 2018.
- [26] R. Berinde, P. Indyk, G. Cormode, and M. J. Strauss, “Space-optimal heavy hitters with strong error bounds,” *ACM Transactions on Database Systems*, vol. 35, no. 4, 2010.
- [27] A. Metwally, D. Agrawal, and A. E. Abbadi, “An integrated efficient solution for computing frequent and top- $k$  elements in data streams,” *ACM Transactions on Database Systems*, vol. 31, p. 1095–1133, sep 2006.
- [28] M. Charikar, K. Chen, and M. Farach-Colton, “Finding frequent items in data streams,” in *International Colloquium on Automata, Languages, and Programming*, pp. 693–703, 2002.
- [29] G. S. Manku and R. Motwani, “Approximate frequency counts over data streams,” in *International Conference on Very Large Databases*, 2002.
- [30] H. Melville, *Moby Dick; Or, The Whale*. Project Gutenberg, 1851.
- [31] L. A. Adamic and B. A. Huberman, “Zipf’s law and the internet.,” *Glottometrics*, vol. 3, no. 1, pp. 143–150, 2002.

- [32] G. Cormode and S. Muthukrishnan, “What’s hot and what’s not: tracking most frequent items dynamically,” *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 249–278, 2005.
- [33] T. Roughgarden and G. Valiant, “Cs168: The modern algorithmic toolbox lecture #2: Approximate heavy hitters and the count-min sketch,” p. 15.
- [34] N. Homem and J. P. Carvalho, “Finding top-k elements in data streams,” *Information Sciences*, vol. 180, no. 24, pp. 4958–4974, 2010.
- [35] R. Real and J. M. Vargas, “The probabilistic basis of jaccard’s index of similarity,” *Systematic biology*, vol. 45, no. 3, pp. 380–385, 1996.

## BIOGRAPHICAL SKETCH

- Nmae plus Research interests
- PhD at UF
- UConn – BS + Goldwater Scholar
- Proof trading