On the Security of the K Minimum Values (KMV) Sketch

Pedro Reviriego, Alfonso Sánchez-Macián, Shanshan Liu, and Fabrizio Lombardi

Abstract—Data sketches are widely used to accelerate operations in big data analytics. For example, algorithms use sketches to compute the cardinality of a set, or the similarity between two sets. Sketches achieve significant reductions in computing time and storage requirements by providing probabilistic estimates rather than exact values. In many applications, an estimate is sufficient and thus, it is possible to trade accuracy for computational complexity; this enables the use of probabilistic sketches. However, the use of probabilistic data structures may create security issues because an attacker may manipulate the data in such a way that the sketches produce an incorrect estimate. For example, an attacker could potentially inflate the estimate of the number of distinct users to increase its revenues or popularity. Recent works have shown that an attacker can manipulate Hyperloglog, a sketch widely used for cardinality estimate, with no knowledge of its implementation details. This paper considers the security of K Minimum Values (KMV), a sketch that is also widely used to implement both cardinality and similarity estimates. Next sections characterize vulnerabilities at an implementation-independent level, with attacks formulated as part of a novel adversary model that manipulates the similarity estimate. Therefore, the paper pursues an analysis and simulation; the results suggest that as vulnerable to attacks, an increase or reduction of the estimate may occur. The execution of the attacks against the KMV implementation in the Apache DataSketches library validates these scenarios. Experiments show an excellent agreement between theory and experimental results.

Index Terms—Data sketches, cardinality, KMV, similarity, security, attack

1 Introduction

Acceleration of big data analytics is a key challenge in large-scale computing systems. As data sets increase in size, simple operations, such as computing the number of distinct elements in a set, become complex [1]. In many applications, accurate estimates are good enough, so designers accelerate many operations by using probabilistic rather than exact algorithms and data structures [2], [3]. These probabilistic data structures are commonly known as sketches [4],[5], [6]. For example, sketches have been proposed for cardinality estimate [7], [8], [9], [10], similarity estimate [11], [12], [13], frequency estimate [14], [15] and membership checking [16], [17] among other operations. Sketches typically reduce both computation and storage dramatically, and thus data processing systems widely utilize them [18], [19].

Most sketches map a dataset to a small pseudorandom set and perform operations (including queries) on this set, rather than executing the operations on the original and significantly larger dataset. While effectively reducing the implementation overhead, it also may allow attackers to manipulate the estimates [20]. The security of streaming algorithms and sampling have been recently analyzed in [21], [22], [23]. For the cardinality estimate, recent works have shown that an attacker can manipulate the estimates of Hyperloglog to produce an estimate that is significantly larger [24] or smaller [25] than the actual cardinality of the data set. This can occur even when the attacker has no knowledge of the implementation details and can just perform user operations on the sketch; for example, an attacker could potentially inflate the estimate of the number of distinct users to increase its revenues or popularity. Therefore, the analysis of the security of sketches to identify potential attacks and countermeasures is important to ensure that their use does not compromise the security of the entire system. Similarly, designers should also consider privacy when using sketches [26].

For applications that perform many operations on the data sets, it is beneficial to use a single sketch for several tasks. One example of such sketches is the K Minimum Values (KMV) sketch that can be used to estimate cardinality, similarity, and also for distinct sampling [9], [10]. To the best of the authors' knowledge, there are no previous studies about the security of the KMV sketch, although it is widely used in data processing applications [27].

This paper analyzes the security of KMV and proposes attacks that increase or reduce its cardinality estimate. These attacks are part of an adversary model that also provides the functionality to manipulate the similarity estimate. We propose two algorithms for the inflation and deflation of the cardinality estimate of KMV; these algorithms characterize vulnerabilities at an implementation-independent level. The feasibility of the attacks is also demonstrated on the KMV implementation of the Apache DataSketches library.

The rest of the paper is organized as follows. Section 2 briefly describes the KMV sketch, the adversarial model considered, and the potential goals of an attacker when manipulating the sketch. Section 3 presents and analyzes the proposed attacks and Section 4 evaluates them both by simulation and testing them on the DataSketches KMV implementation. The paper ends in Section 5 with the conclusion and a discussion of topics and ideas of interest for future work.

2 Preliminaries

This section initially describes the KMV sketch. Then it presents the considered adversarial model, and finally, it describes some manipulations that an attacker may want to perform on the KMV.

2.1 K Minimum Values

The K Minimum Values (KMV) sketch is conceptually very simple; for any element x in the stream, its hash function $h(x)$ is computed and the K lowest or minimum values of $h(x)$: $m_1, m_2, \dots, m_K$ are kept. This is illustrated in Fig. 1. To insert an element x, first $h(x)$ is computed (a so-called "salt" can be added to the hashing process to force its uniqueness if needed), and then it is compared to the maximum ($maxK$) of the stored K minimum values; if it is the same or larger, no update is required. Otherwise, x is added to the list of KMVs if it does not already exist, and the corresponding $h(x)$ is used to update $maxK$ (i.e., the previous maximum of the K minimum values is removed). Since the first scenario occurs frequently, the insertion requires, in most cases, just a hash computation and a comparison. Once elements are inserted, both cardinality and similarity can be estimated based on the K values (i.e., $m_1, m_2, \dots, m_K$).

To estimate the cardinality, the following equation is used:

$$C_{KMV} = \frac{K-1}{maxK}, \quad (1)$$

where $maxK$ is the maximum value of the KMVs (mapping the range of values of $h(x)$ to [0, 1]). This estimate is unbiased and has a relative accuracy (standard error) [1] of:

1. See Equation 3.14 in https://github.com/apache/datasketches-website/blob/master/docs/pdf/ThetaSketchEquations.pdf

• Pedro Reviriego is with the Universidad Carlos III de Madrid, Leganés 28911, Madrid, Spain. E-mail: revirieg@it.uc3m.es.
• Alfonso Sánchez-Macián is with the ARIES Research Center, Universidad Antonio de Nebrija, 28040, Madrid, Spain. E-mail: asanchep@nebrija.es.
• Shanshan Liu and Fabrizio Lombardi are with the Northeastern University, Department of ECE, Boston, MA 02115 USA. E-mail: {ssliu, lombardi}@ece.neu.edu.

Manuscript received 16 Feb. 2021; revised 16 July 2021; accepted 27 July 2021. Date of publication 2 Aug. 2021; date of current version 2 Sept. 2022.
(Corresponding author: Pedro Reviriego.)
Digital Object Identifier no. 10.1109/TDSC.2021.3101280

1545-5971 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Xiamen University. Downloaded on February 26,2024 at 12:51:27 UTC from IEEE Xplore. Restrictions apply.

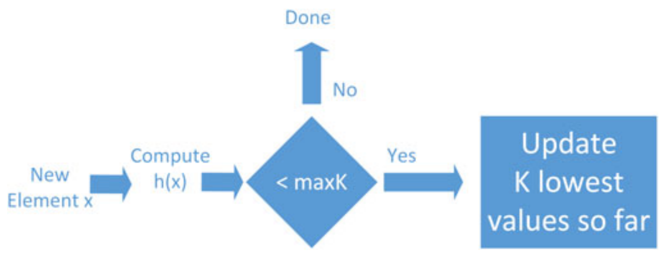

Fig. 1. Illustration of the KMV calculation.

$$\sigma < \frac{1}{\sqrt{K-2}}, \tag{2}$$

This procedure is used unless fewer than K elements have been inserted in the sketch. In this case, the algorithm returns the exact number of elements, because the list of K minimum values is not full, and it can be easily computed.

An interesting feature of the KMV sketch is that the cardinality of the union of the sets can be estimated from their KMVs if the same hash function $h(x)$ (and salt if applied) has been used for both sets. This is a key feature in several applications that independently compute estimates on many subsets, because it enables merging of the KMVs.

To estimate the similarity of two sets, A and B, for which the KMVs have been computed, the number of matching KMVs $(m_i(A) = m_j(B))$ between them is computed and divided by K to obtain an estimate of the Jaccard similarity coefficient. As discussed previously, this requires the use of the same hash function $h(x)$ and salt when computing the KMVs of both sets.

Finally, if one or more parameters are associated with each of the KMVs, designers can use the sketch to estimate the distribution of those parameters among the different values and perform more complex operations [27].

2.2 Adversarial Model

The manipulation of the cardinality or similarity estimates of a KMV sketch is trivial if the used hash function $h(x)$ and salt are known to the attacker. In this case, the attacker can test elements to check their value of $h(x)$ and select the ones that meet the attack requirements. For example, if the attacker wants to inflate the cardinality estimate, it can simply select K elements with very small values of $h(x)$ and then insert them into the KMV sketch. Conversely, if the attacker wants to create a large data set that produces a small cardinality estimate, it can simply select elements with large values of $h(x)$. Similarly, if the attacker wants to make sure that a given data set is not detected as similar to another set, it can just add K elements with values of $h(x)$ smaller than those in the original set. However, in many settings, the attacker may not have access to the hash function $h(x)$ and salt and thus, the simple attacks just described are not possible.

This paper uses a more realistic black-box adversary model; it assumes that the attacker has only access to the sketch operations, but not to the sketch implementation details, such as for example, the used hash functions, or the salt. This means that the attacker can create sketches, insert elements into them and ask for the cardinality estimate. This is the functionality that many sketch implementations expose to users. A key observation is that to support merging and similarity estimate for different instances of KMV sketches, they must use the same hash function $h(x)$ and salt. This in turn means that a given set produces the same estimate on different KMV instances and thus the attacker can generate an attack set on one instance and use it on a different one. This black-box adversarial model is the same as the one used in recent studies on the security of Hyperloglog [25].

2.3 Manipulation of KMV

An attacker may have different reasons to manipulate the KMV sketch. For example, if KMV is used to estimate the number of different users that access a service (such as reproducing a video), the attacker may want to inflate the estimates to make some videos appear as popular or even to increase revenue for the content creators if they are paid according to the number of users. A different scenario is that, when KMV is used to estimate the cardinality of node connections to detect Distributed Denial of Service (DDoS) attacks [28], the attacker may want to use source addresses that do not increase the cardinality estimate, so that the DDoS is not detected by the KMV [25]. These two simple examples show the scenarios by which an attacker may benefit from being able to either inflate or deflate the cardinality estimate of the KMV sketch.

Another example occurs when using sketches to detect if a document is similar to another [29]. In this case, the attacker may want to change the similarity estimate so that similar documents are not detected. In particular, this can be achieved by adding a few additional elements to the document, so that even though its true similarity remains high, the estimate result is reduced. Conversely, the attacker may also want to build a set that is different from another set but for which the sketch produces a high similarity estimate.

Finally, if KMV is used to sample elements, the attacker may want to manipulate the sampling process, so that elements of choice are selected and thus, the sample does not correspond to the real distribution of the set.

3 Attacks

This section first presents attacks to inflate and deflate the KMV cardinality estimate for the adversarial model described in the previous section. Then, it discusses attacks to the similarity estimate, as they are based on the inflation and deflation cardinality attacks. It also analyzes the proposed attacks by considering an ideal implementation of the KMV sketch in which the hash functions behave as uniformly distributed random functions, and no rounding (or quantization) is used when storing (or processing) the values.

3.1 Inflation Attack

As an example, consider an attacker that wants to inflate the count for distinct users accessing a video, so that it reaches one million and the count is estimated using a KMV sketch. Then, to inflate the cardinality estimate, the attacker must find a small set A with cardinality C_A that produces an estimate $C_{KMV} = 10^6$, such that $C_{KMV} \gg C_A$ (e.g., $C_A = 10^4$). More generally, for the attack to be effective the size of the attack set C_A should be at least one order of magnitude smaller than the desired size C_{KMV}. This can be accomplished by starting with a set of the desired size C_{KMV} and inserting it into an empty KMV. The cardinality estimate is computed before and after inserting each element; when the estimates are different, the element is added to the initial attack set I. Its principle is that this will identify elements with small values of $h(x)$, because only those (that update the KMVs) modify the cardinality estimate. However, this set can still include a large number of elements, because to reach the final KMVs many intermediate updates may be required. To reduce the size of the attack set, the attacker can perform a second iteration by creating another empty sketch and repeating the process, but this time inserting the initial attack set I in reverse order. The insight is that elements inserted at the end of the initial attack set tend to have smaller values of $h(x)$ because they updated the KMVs after many elements have been inserted. Therefore, by repeating the process with the initial attack set in reverse order, the attack set can be reduced, while achieving the same cardinality estimate. Algorithm 1 formally presents the algorithm for the attack; this process is implementation independent.

Authorized licensed use limited to: Xiamen University. Downloaded on February 26,2024 at 12:51:27 UTC from IEEE Xplore. Restrictions apply.

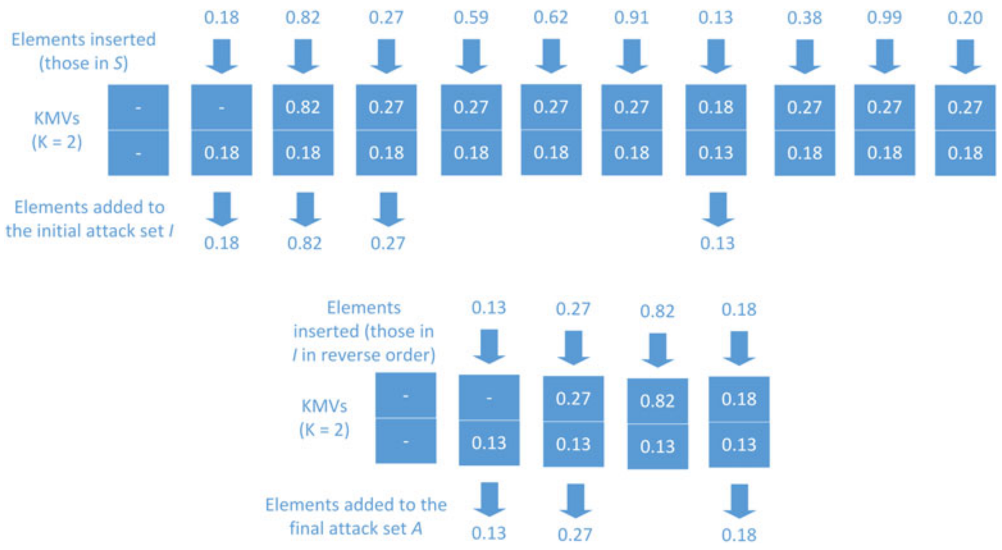

Fig. 2. Example of the inflation attack: building the initial set (top) and the final set (bottom).

Algorithm 1. Attack to Inflate the Cardinality Estimate of KMV

```
Input: Set $S$ with the desired cardinality $C$
Output: Attack set $A$
1: Create an empty KMV sketch $KMV_1$
2: Create an empty set $I$
3: for $s \in S$ do
4:     $C_{pre}$ = Estimate($KMV_1$)
5:     Insert $s$ in $KMV_1$
6:     $C_{post}$ = Estimate($KMV_1$)
7:     if $C_{post} > C_{pre}$ then
8:         Add $s$ to $I$
9:     end if
10: end for
11: Create an empty KMV sketch $KMV_2$
12: Create set $RI$ with the elements of $I$ in reverse order
13: Create an empty set $A$
14: for $s \in RI$ do
15:     $C_{pre}$ = Estimate($KMV_2$)
16:     Insert $s$ in $KMV_2$
17:     $C_{post}$ = Estimate($KMV_2$)
18:     if $C_{post} > C_{pre}$ then
19:         Add $s$ to $A$
20:     end if
21: end for
```

To illustrate the process, Fig. 2 shows an example of a set S with ten elements with $h(x)$ values of 0.18, 0.82, 0.27, 0.59, 0.62, 0.91, 0.13, 0.38, 0.99, 0.20, in which $K = 2$. Then, the minimum values are 0.13 and 0.18 and the cardinality estimate is 1/0.18 = 5.55. When the elements of S are inserted in order in the KMV, the ones that update the $K = 2$ minimum values are the two first elements: 0.18, 0.82 and then 0.27, 0.13. These four elements make up the initial attack set I (that is already significantly smaller than S). When the elements of I are inserted in reverse order, only 0.13, 0.27, 0.18 update the KMV and thus, they are added to the final attack set A (which is smaller than I).

Next paragraphs analyze the expected size of the sets I and A obtained when executing Algorithm 1.

As discussed previously, when fewer than K elements have been inserted in the sketch, the Algorithm returns the exact number of elements. Therefore, the first K elements added to the KMV increase the cardinality estimate and thus they are added to the set I.

To estimate the number of remaining $C - K$ elements in S to be inserted in I, consider the insertion of the zth element in S. Then, the expectation of the maximum of the K minimum values $maxK$ is given by:

$$\mathbb{E}[maxK] = \frac{K-1}{z-1}. \tag{3}$$

Therefore, the probability that the zth element is smaller than $maxK$ (so that it is added to I), is in the expectation $\frac{K-1}{z-1}$. Therefore, in expectation, the number of elements in I is given by:

$$\mathbb{E}[|I|] \approx K + \sum_{i=K+1}^{C} \frac{K-1}{i-1}, \tag{4}$$

For large values of C and K, the second term can be approximated by $K \cdot \log(\frac{C}{K})$ and thus:

$$\mathbb{E}[|I|] \approx K + K \cdot \log(\frac{C}{K}). \tag{5}$$

This analysis shows that the set I has a size of $O(\log(C))$, so already significantly smaller than S when C is large. Recall that for an ideal implementation, I generates the same cardinality estimate as S on the KMV.

Now analyze the size of the final attack set A. Since the elements are added in reverse order, the first K elements added to the new KMV are the last ones added to the set I. After removing these last K elements, the set I has in expectation a size of:

$$\begin{aligned} K + K \cdot \log(\frac{C}{K}) - K &= K + K \cdot \left(\log(\frac{C}{K}) - 1\right) \\ &= K + K \cdot \left(\log(\frac{C}{K}) - \log(e)\right) = K + K \cdot \log(\frac{\frac{C}{e}}{K}), \end{aligned} \tag{6}$$

which corresponds to a set of size $\frac{C}{e}$ where e is Euler's number.

Therefore, once the first K elements have been added, only those that correspond to the first $\frac{C}{e}$ elements in S, can modify the KMVs and be added to set A. As the other $C - \frac{C}{e}$ elements have added K elements to I, the first $\frac{C}{e}$ elements add in expectation $K \cdot \frac{\frac{C}{e}}{C - \frac{C}{e}} = K \cdot \frac{1}{e-1}$ elements to A so that its total size is in expectation:

$$\mathbb{E}[|A|] = K + K \cdot \frac{1}{e-1} = K \cdot \frac{e}{e-1}, \tag{7}$$

which is approximately $1.58 \cdot K$, so a very small attack set. The attack set can be further reduced by shuffling A and taking it as the initial set S and repeating the process as discussed previously.

Authorized licensed use limited to: Xiamen University. Downloaded on February 26,2024 at 12:51:27 UTC from IEEE Xplore. Restrictions apply.

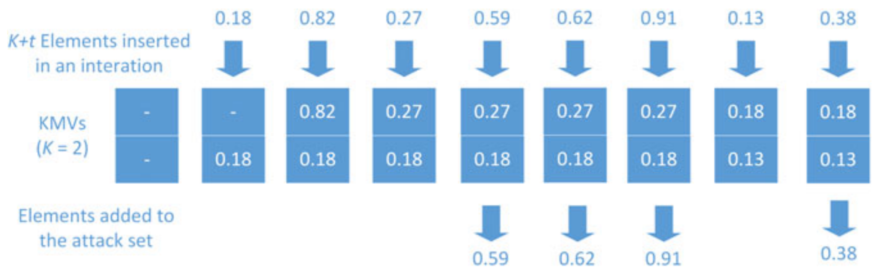

Fig. 3. Example of the deflation attack.

However, the benefit of this process is small because A has already a small number of elements and it can only be reduced to K. Recall that K is a lower bound for the attack set size, because when inserting fewer than K distinct elements, the KMV returns the exact cardinality and thus the manipulation is not possible.

3.2 Deflation Attack

Another scenario arises when the attacker wants to reduce the KMV estimate. For example, consider a Distributed Denial of Service (DDoS) attack for which the attacked network monitors the number of external IP addresses that generate the incoming traffic to detect the attack. Then, if the attacker can identify many IP addresses that produce a small KMV estimate, detection will likely fail. For example, if it is possible to build a set of one hundred thousand IP addresses for which the KMV estimate is one thousand, then the attack will most likely not be detected; in general, it seems that the reduction in the KMV estimate should be at least of an order of magnitude to make the attack useful.

Algorithm 2. Attack to Deflate the Cardinality Estimate of KMV

```
Input: Parameter $t$, large set of distinct elements $S$ and
       desired cardinality of the attack set $C$
Output: Attack set $A$
1:  Create an empty set A
2:  Create an empty KMV sketch $KMV_t$
3:  $j = 0$
4:  $inc = false$
5:  for $s \in S$ do
6:    if $inc$ then
7:      $j = j + 1$
8:    end if
9:    $C_{pre}$ = Estimate($KMV_t$)
10:   Insert $s$ in $KMV_t$
11:   $C_{post}$ = Estimate($KMV_t$)
12:   if $C_{post} = C_{pre}$ then
13:     $inc = true$
14:     Add $s$ to $A$
15:     if (size(A) = C) then break;
16:     end if
17:   end if
18:   if $(j \geq t)$ then
19:     Create an empty KMV sketch $KMV_t$
20:     $j = 0$
21:     $inc = false$
22:   end if
23: end for
```

To reduce the KMV estimate, the attacker needs to find elements that have large values of $h(x)$. To do so, the attacker can create an empty KMV, insert distinct elements and take the first ones that do not increase the cardinality estimate for the attack set A. Those elements would necessarily have large values of $h(x)$ as otherwise they would increment the KMV cardinality estimate. However, as elements are inserted, the values of $h(x)$ that do not increase the estimate, reduce; so, to build a large attack set, the attacker needs to create another empty KMV and repeat the process again. In more detail, the process starts and once the first element that does not increase the cardinality estimate, is inserted, a counter is started and incremented on each subsequent element insertion, until it reaches a configurable value t. At that point, a new empty KMV is created and the process starts again. This is repeated until the desired number of elements has been added to the attack set A.

Algorithm 2 formally describes the attack algorithm (also implementation independent). For each element inserted, the Algorithm checks if the KMV estimate is incremented; if it does not, the element is added to the attack set. Additionally, when this occurs for the first time (which should be soon after K insertions), the counter for the remaining t insertions in the round is started (by setting inc to true). This is performed, so that the Algorithm executes with no knowledge of the value K used in the KMV under attack. At the end of each round a new KMV is constructed, and the process is repeated until an attack set with the desired size is built. In an ideal KMV implementation, each iteration inserts approximately $K + t$ elements into KMV as the first K elements always increase the cardinality estimate as discussed previously.

To illustrate the process, Fig. 3 shows another example of a set S with eight elements with $h(x)$ values of 0.18, 0.82, 0.27, 0.59, 0.62, 0.91, 0.13, 0.38, in which $K = 2$ and $t = 6$. In this case, the values that do not update the KMVs and are added to the attack set A when inserting S are 0.59, 0.62, 0.91, 0.38. Therefore, these elements tend to produce a small cardinality estimate because their values are large.

Elements inserted into the set must be above $maxK$ when the KMV estimate is below approximately $K + t - 1$ (as after inserting $K + t$ elements, a new KMV sketch is created for the next group of elements); therefore, they should have hash values $h(x)$ larger than $\frac{K-1}{K+t-1}$. This means that the built attack set A generates a KMV estimate of at most $K + t - 1$. More formally, for a KMV sketch in which $K + t$ or fewer elements have been inserted, $maxK$ will be larger than $\frac{K-1}{K+t} - \epsilon$ with high probability for any ϵ significantly larger than $\frac{1}{\sqrt{K-2}}$. The proof is straightforward, because we are just stating that the KMV estimate will be (with high probability) close to its expectation namely, within distance ϵ that is larger than the standard deviation of the estimate. Finally, in most applications $K \gg 1$, so ϵ can be neglected when t is small.

Therefore, the attacker can ensure that the estimated cardinality is below a given target value by appropriately selecting the value of t. As K is a lower bound for the cardinality estimate of the attack set, then it is likely that such value is close to the lowest possible value by using a small value for t. However, the larger the t, the fewer elements (and KMV constructions) need to be tested to build the attack set. In more detail, for each iteration of approximately $K + t$ elements, the expected number of elements added to the attack set A can be approximated by:

$$t - \sum_{i=K+1}^{K+t} \frac{K-1}{i-1} \approx t - K \cdot \log\left(\frac{K+t}{K}\right). \quad (8)$$

Authorized licensed use limited to: Xiamen University. Downloaded on February 26,2024 at 12:51:27 UTC from IEEE Xplore. Restrictions apply.

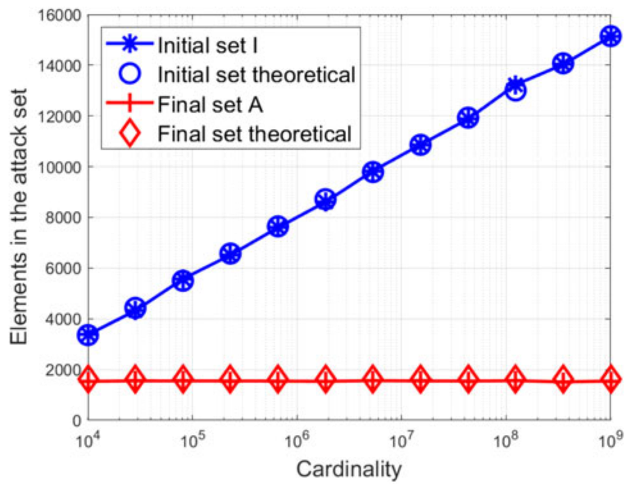

Fig. 4. Inflation Attack: Attack set size versus KMV cardinality estimate for the initial set I and the final set A.

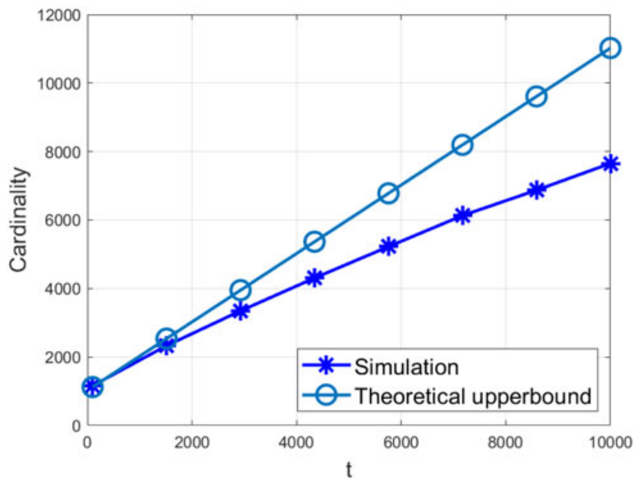

Fig. 5. Deflation Attack: KMV cardinality estimate of the attack set A with size 100,000 versus t.

So, the fraction of elements inserted into the attack set can be computed as:

$$\frac{t - K \cdot \log\left(\frac{K+t-1}{K}\right)}{K+t}, \tag{9}$$

which increases towards one as t gets significantly larger than K.

3.3 Similarity Attacks

To make a set D (that is similar to a set E) to have a low similarity estimate in the KMV sketch, the attacker just needs to insert elements in D that modify its KMVs (as these are compared with E to estimate the similarity). Therefore, the attacker requires a set of elements A that have low values of $h(x)$. This set is precisely the previously described attack set A in the inflation attack of Section 3.1. Therefore, by selecting C formed by elements not in E and larger than $|D|$ and running the inflation attack, the attacker will get a set of size approximately $1.58 \cdot K$ that when added to D, reduces its similarity estimate with E.

Therefore, assume that the attacker wants to make sets D and E (that initially are different) to be similar in the KMV estimate; then the attacker executes an inflation attack on set E and adds the resulting attack set A to D. This produces a similarity estimate for KMV that corresponds to the similarity of $D \cup E$ and E. If the attacker manipulates both sets, the same operation is performed by running the inflation attack on set D and adding the resulting attack set A to E. In this case, the similarity of the modified sets D and E is close to one because both sets have estimates similar to those of $D \cup E$.

As per the previous discussion, the cardinality estimate attacks produce also the results required to manipulate the similarity estimate in the KMV sketch.

4 Evaluation

This section first validates the theoretical analysis on an ideal KMV implementation and then, it shows its feasibility in a publicly available and widely used software library. In all experiments $K = 1024$, but the results were similar for other values of K and are omitted for brevity.

4.1 Initial validation

To validate the analysis and approximations presented in the previous section, this work implements and tests an ideal KMV sketch.[2] This ideal implementation does not round the cardinality estimate, and returns the fractional value, using a large precision in all operations. Therefore, any change on the KMVs affects the cardinality estimate. The proposed attacks have been run in this ideal environment.

4.1.1 Inflation Attack

For the inflation attack, Algorithm 1 has been run for sets S of different cardinality C and $K = 1024$. Then, the implementation inserted the generated initial attack set I and the final attack set A into an empty KMV to check that they produce the same cardinality estimate as S. After verifying this, the size of the attack sets is compared to the theoretical estimates given by Equations (5) and (7). Fig. 4 shows the results and plots the attack set size (y-axis) needed to achieve a given target cardinality estimate in the KMV (x-axis). This plot captures the effectiveness of the attack. As shown in Fig. 4, the results match the theoretical estimates and the final attack set size is almost constant and does not depend on the size of S. This confirms that the attack can produce an arbitrarily large cardinality estimate with a small number of elements.

4.1.2 Deflation Attack

The deflation attack has been simulated for different values of t with $K = 1024$ and for an attack set size of 100,000 elements. Fig. 5 summarizes the results showing the cardinality estimate of the KMV for the attack set size versus t. The smaller the estimate, the more effective the attack. The theoretical upper bound $K + t$ is also plotted. As per Fig. 5, the attack set produces an estimate that is below the upper bound as predicted by the previous analysis. This is because the K minimum values of the attack set tend to be above $\frac{K-1)}{(K-t-1)}$. Again, the attacker can control the KMV estimate, so to make it significantly smaller than the true cardinality of the attack set.

Finally, Fig. 6 shows the fraction of elements added to the attack set in each round of $K + t$ elements; it also shows the value of Equation (9). The simulation results match the theoretical approximation; as t increases, so also the fraction of elements added to the attack set increases. Therefore, the use of larger values of t requires fewer iterations to build the attack set (as discussed in the previous section).

4.2 Feasibility validation

After validating the theoretical analysis in an ideal setting, a script ran the proposed attacks on the KMV implementation of the Apache DataSketches library [27]; this is part of the Theta sketch framework in which KMV is a particular case [19]. The implementation of KMV (using the Theta Sketch QuickSelect algorithm) is slightly different

2. The code for the attacks presented in this subsection and the next one are available in link https://github.com/amacian/KMVattack

Authorized licensed use limited to: Xiamen University. Downloaded on February 26,2024 at 12:51:27 UTC from IEEE Xplore. Restrictions apply.

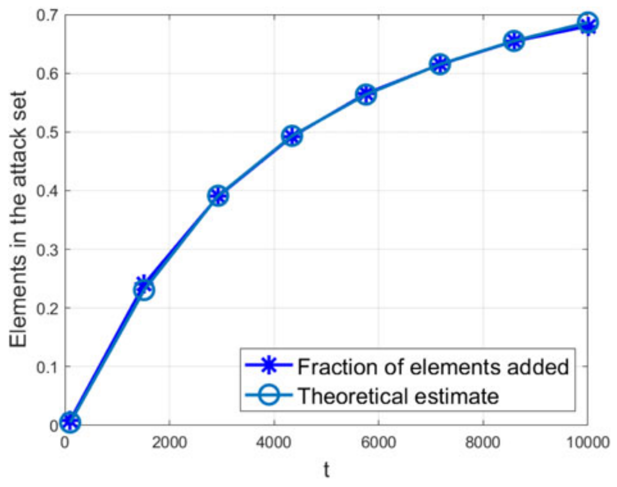

Fig. 6. Deflation Attack: Fraction of elements added to the attack set on each iteration.

from the traditional version, because the number of minimum values kept is not fixed (i.e., it varies between K and $2K$ to support other sketches, like the Theta sketch, that need to resize the structure dynamically). Therefore, results are expected to deviate from the theoretical analysis presented in the previous section. This implementation uses a high-performing 128-bit MurmurHash3 hash function, that behaves well in terms of distribution, avalanche behavior and collision resistance. In any case, the goal of this paper is to show that the proposed attacks can be utilized against real implementations of the sketch used in commercial systems. Next subsections presents and discusses the results for the inflation and deflation attacks on the DataSketches implementation.

4.2.1 Inflation Attack

A Java program ran the attack on the Java API Core 1.3.0-incubating version for the same cardinalities evaluated in the previous section (ranging from ten thousand to one billion distinct elements). The experiment kept the default values for the UpdateSketchBuilder API element unaltered, except for the value of the nominal entries, that it set to 1024 (among other values). Randomly generated elements of type double from a uniform distribution have been used to build the original set S. Subsets have been extracted to create the reduced attack sets. The first experiment tested the attacks to check that the constructed sets (the initial set I and the final set A) generated a similar cardinality estimate for the sketch as the initial set S. Fig. 7 shows the results for one run; the results show that the cardinality estimates are very similar and the difference is smaller than 5 percent in all cases. Similar values have been observed in other runs. The small variations on the cardinality estimates may be due to the DataSketches KMV that uses a dynamic value of K as well as other implementation details. Nevertheless, these results confirm the feasibility of the attack.

Next paragraphs evaluate the sizes of the attack sets I and A. Fig. 8 reports the average across ten runs. The sizes of both the initial set I and the final set A are larger than the theoretical estimates. The values are on average 1.32x (1.40x) larger than the theoretical ones for the initial (final) attack set. As discussed previously, this trend is expected, because the DataSketches implementation uses a value of K that is dynamic between K and $2K$.

4.2.2 Deflation Attack

The deflation attack has been tested using the same values of t as for the initial validation (i.e. 100, 1514, 2929, 4343, 5757, 7171, 8586 and 10000) and targeting an attack set A of size 100,000 elements. The experiment originally generated sets S of 10 million of uniformly

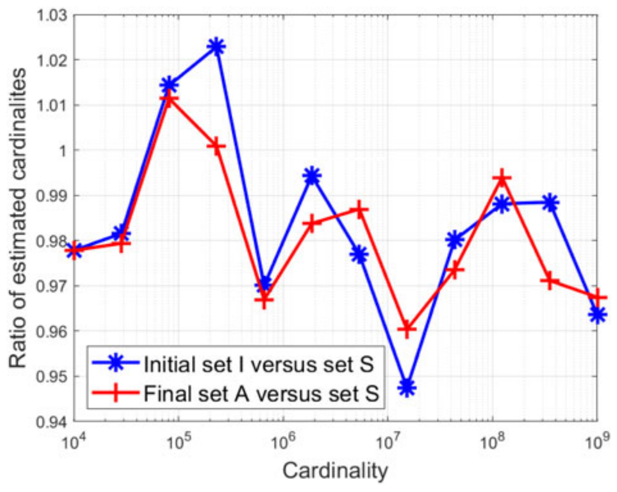

Fig. 7. Inflation Attack on DataSketches: Ratio of the cardinality estimates for the initial set I and the final set A versus the estimate for the original set S.

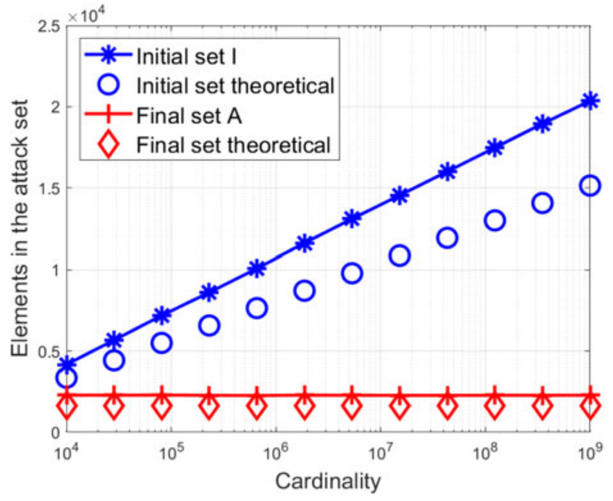

Fig. 8. Inflation Attack on DataSketches: Attack set size versus KMV cardinality estimate for the initial set I and the final set A.

distributed randomly generated double values to obtain the attack set A. Fig. 9 shows the cardinality estimate of the KMV for the attack set, also plotting the theoretical upper bound $K + t$. As per Fig. 9, the KMV estimated cardinality for the attack set A is in some cases slightly above the theoretical upper bound of $K + t$. This again can be explained as the DataSketches implementation uses a dynamic K that can be larger than the nominal value. Also, in this scenario, the attack is capable of generating a large set that has a low cardinality estimate (controlled by the attacker using the parameter t).

Fig. 10 shows the fraction of elements that are added to the attack set in each round of insertions; it also shows the theoretical value of Equation (9). Results follow the same trends as those of Equation (9) with some differences that can again be attributed to the DataSketches implementation.

5 Conclusion and Future Work

This paper has studied the security of the KMV sketch and it has proposed, analyzed and evaluated several attacks based on implementation-independent vulnerabilities. The results have shown that without knowing its implementation details, an attacker can manipulate the sketch to increase or reduce the cardinality and similarity estimates. This finding is of critical importance because many data processing applications use KMV.

Authorized licensed use limited to: Xiamen University. Downloaded on February 26,2024 at 12:51:27 UTC from IEEE Xplore. Restrictions apply.

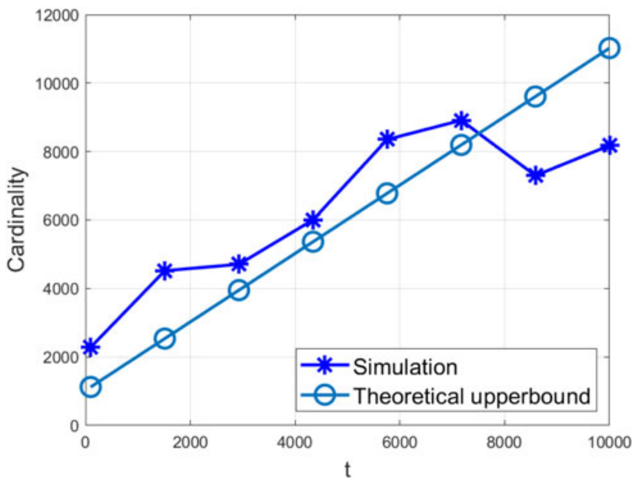

Fig. 9. Deflation Attack on DataSketches: KMV cardinality estimate of the attack set A with size 100,000 versus t.

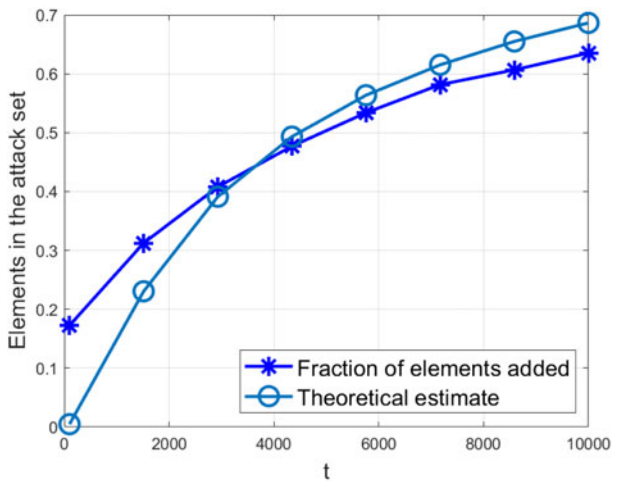

Fig. 10. Deflation Attack on DataSketches: Fraction of elements added to the attack set on each iteration.

Future work will concentrate on investigating potential countermeasures that the designers of data processing systems can employ to prevent or at least detect the attacks. A simple solution is to use a different random salt on each KMV instance, so that the attack set built on one KMV instance does not work on a different instance. However, in this case, the instances can no longer be combined to compute the cardinality of the unions and used for the similarity estimate. This significantly reduces the functionality of the sketch and may not be acceptable in many applications. A more sophisticated (and costly) approach is to use an auxiliary KMV for each instance that uses a random salt as proposed in [25]. In this case, each KMV instance has two cardinality estimates: the original one and the salted one. Under normal operation, both cardinality estimates are similar so when they diverge, an attack is detected. The main drawback of this strategy (denoted as Salted Not Salted (SNS)) is that additional storage and computational effort is required for the auxiliary salted estimator. Alternatively, rounding the output values produced by the sketch (as proposed in [21]) will also make the attack harder by limiting the ability of the attacker to identify the desired elements for the attack set. More broadly, it is of interest and relevance to investigate more efficient methods to detect these attacks.

Acknowledgments

Pedro Reviriego acknowledges the support of the ACHILLES project PID2019-104207RB-I00 and the Go2Edge network RED2018-102585-T funded by the Spanish Ministry of Economy and Competitivity and of the Madrid Community research project TAPIR-CM under Grant P2018/TCS-4496.

References

[1] M. Honarkhah and A. Talebzadeh, “Hyperloglog in presto: A significantly faster way to handle cardinality estimation,” 2018. [Online]. Available: https://engineering.fb.com/2018/12/13/data-infrastructure/hyperloglog/
[2] L. Blog, “Data matters,” 2013. [Online]. Available: https://looker.com/blog/practical-data-science-amazon-announces-hyperloglog
[3] M. Azure, “Documentation,” 2013. [Online]. Available: https://docs.microsoft.com-/en-us/azure/kusto/query/dcount-aggfunctionestimation-accuracy
[4] K. Beyer, R. Gemulla, P. J. Haas, B. Reinwald, and Y. Sismanis, “Distinct-value synopses for multiset operations,” *Commun. ACM*, vol. 52, pp. 87–95, Oct. 2009.
[5] G. Cormode, “Data sketching,” *Commun. ACM*, vol. 60, Aug. 2017, Art. no. 48–55.
[6] J. Tang, Y. Cheng, Y. Hao, and W. Song, “Sip flooding attack detection with a multi-dimensional sketch design,” *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 6, pp. 582–595, Nov./Dec. 2014.
[7] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, “Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm,” in *Proc. Int. Conf. Anal. Algorithms*, 2007, pp. 137–15.
[8] O. Ertl, “New cardinality estimation algorithms for HyperLogLog sketches,” *CoRR*, vol. abs/1702.01284, 2017.
[9] Z. Bar-Yossef , T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, “Counting distinct elements in a data stream,” in *Proc. 6th Int. Workshop Randomization Approximation Techn.*, 2002, Art. no. 110.
[10] F. Giroire, “Order statistics and estimating cardinalities of massive data sets,” *Discrete Appl. Math.*, vol. 157, no. 2, pp. 406–427, 2009.
[11] A. Broder, “On the resemblance and containment of documents,” in *Proc. Compression Complexity SEQUENCES 1997 (Cat. No.97TB100171)*, 1997, pp. 21–29.
[12] O. Ertl, “ProbMinHash – a class of locality-sensitive hash algorithms for the (probability) jaccard similarity,” *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2020.3021176.
[13] A. Shrivastava, “Simple and efficient weighted minwise hashing,” in *Proc. Advances Neural Inf. Process. Syst.*, 2016, pp. 1498–1506.
[14] G. Cormode and M. Hadjieleftheriou, “Methods for finding frequent items in data streams,” *VLDB J.*, vol. 19, pp. 3–20, Feb. 2010.
[15] M. Charikar, K. Chen, and M. Farach-Colton , “Finding frequent items in data streams,” *Theor. Comput. Sci.*, vol. 312, pp. 3–15, Jan. 2004.
[16] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, “Optimizing Bloom filter: Challenges, solutions, and comparisons,” *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1912–1949, Second Quarter 2019.
[17] M. Mitzenmacher, S. Pontarelli, and P. Reviriego, “Adaptive cuckoo filters,” *ACM J. Exp. Algorithmics*, vol. 25, pp. 1–20, 2020.
[18] S. Heule, M. Nunkesser, and A. Hall, “Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm,” in *Proc. Int. Conf. Extending Database Technol.*, 2013, pp. 683–692.
[19] A. Dasgupta, K. J. Lang, L. Rhodes, and J. Thaler, “A framework for estimating stream expression cardinalities,” in *Proc. 19th Int. Conf. Database Theory*, 2016, pp. 6:1–6:17. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2016/5775, doi: 10.4230/LIPIcs.ICDT.2016.6.
[20] B. K. Samanthula and W. Jiang, “Secure multiset intersection cardinality and its application to jaccard coefficient,” *IEEE Trans. Dependable Secure Comput.*, vol. 13, no. 5, pp. 591–604, Sep./Oct. 2016.
[21] O. Ben-Eliezer , R. Jayaram, D. P. Woodruff, and E. Yogev, “A framework for adversarially robust streaming algorithms,” in *Proc. 39th ACM SIGMOD-SIGACT-SIGAI Symp. Princ. Database Syst.*, 2020, pp. 63–80.
[22] D. P. Woodruff and S. Zhou, “Tight bounds for adversarially robust streams and sliding windows via difference estimators,” *CoRR*, vol. abs/2011.07471, 2020.
[23] O. Ben-Eliezer and E. Yogev, “The adversarial robustness of sampling,” in *Proc. 39th ACM SIGMOD-SIGACT-SIGAI Symp. Princ. Database Syst.*, 2020, pp. 49–62.
[24] P. Reviriego, P. Adell, and D. Ting, “HyperLogLog (HLL) security: Inflating cardinality estimates,” Nov. 2020.
[25] P. Reviriego and D. Ting, “Security of HyperLogLog (HLL) cardinality estimation: Vulnerabilities and protection,” *IEEE Commun. Lett.*, vol. 24, no. 5, pp. 976–980, May 2020.
[26] D. Desfontaines, A. Lochbihler, and D. A. Basin, “Cardinality estimators do not preserve privacy,” in *Proc. Priv. Enhancing Technol.*, 2019, pp. 26–46. [Online]. Available: https://doi.org/10.2478/popets-2019-0018
[27] Apache, “Apache datasketches: A software library for stochastic streaming algorithms,” 2021. [Online]. Available: http://datasketches.apache.org/
[28] Y. Liu, W. Chen, and Y. Guan, “Identifying high-cardinality hosts from network-wide traffic measurements,” *IEEE Trans. Dependable Secure Comput.*, vol. 13, no. 5, pp. 547–558, Sep./Oct. 2016.
[29] A. Gudkov, “Efficient implementation of minhash,” 2019. [Online]. Available: http://gudok.xyz/minhash1/

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: Xiamen University. Downloaded on February 26,2024 at 12:51:27 UTC from IEEE Xplore. Restrictions apply.