Analyzing and Assessing Pollution Attacks on Bloom Filters: Some Filters are More Vulnerable than Others

Pedro Reviriego
Universidad Carlos III de Madrid
Madrid, Spain
revirieg@it.uc3m.es

Ori Rottenstreich
Technion
Haifa, Israel
or@technion.ac.il

Shanshan Liu and Fabrizio Lombardi
Northeastern University
Boston, USA
sliu@ece.neu.edu, lombardi@ece.neu.edu

***Abstract*—Bloom filters are probabilistic data structures that are popular in networking for set representation; however, they show an inherent inaccuracy due to false positives. One of the potential attacks on Bloom filters is to pollute them with elements that cause the filter to have a larger false positive probability than under normal operation; Pollution is simple when an attacker knows the details of the filter implementation. Recent research has shown that also black-box adversaries can pollute a counting Bloom filter (a common variant of the filter that also supports removals) with no knowledge of its implementation. As over time, many variants and improvements of Bloom filters have been proposed, it is of interest to study whether they can also be polluted and if so also the increase in their false positive probability. This paper first proposes and then evaluates pollution attacks for some of the most common variants including the Block Bloom filters (BBFs), the Variable Increment and Fingerprint Counting Bloom filters (VI-CBFs and FP-CBFs). The results show that with or without knowledge of the implementation, these variants of the Bloom filter are significantly more vulnerable to pollution attacks than the traditional Bloom filter. In particular, BBFs are extremely vulnerable, so providing an insight on their impact and use in practical systems when the number of memory accesses per lookup must be reduced.**

***Index Terms*—Bloom Filters; Security; Pollution attacks; Black-box adversaries.**

I. Introduction

Security is a major issue for computing and networking. As systems become more sophisticated, attackers try to find vulnerabilities on useful algorithms, for example for denial of service [1], [2]. Probabilistic data structures or sketches are increasingly used to speed up data processing [3] and thus, they have become a potential target for attackers [4]–[6]. Among these probabilistic structures, Bloom filters [7], [8] implement set representations and have been utilized in numerous applications, such as packet classification [9], spam filtering [10], [11], distributed systems [12], blockchain [13], [14] and beyond networking. Bloom filters answer membership queries, but with false positives at a probability that is often kept low and controlled by the allocated memory size [15].

This has motivated multiple studies on Bloom filter security and different attacks have been proposed [16]. An attack class of particular interest is the so-called pollution attacks that were described as a tool to degrade the filtering effectiveness by carefully selecting the items for insertion [17]. In these studies, it is assumed that an attacker has access to the filter implementation details (an assumption that happens only rarely in many practical scenarios). However, a recent work has shown that pollution attacks can be applicable also when an adversary has no knowledge of the filter implementation details [18].

Bloom filters have been enhanced [19] to extend functionality and improve accuracy; for example, the Counting Bloom filters have been proposed to support the removal of elements using counters. More recently, these counters have been used to improve performance by using variable increments [20] or fingerprints [21]. Another improvement is the use of blocks to improve locality and reduce the number of memory accesses needed to perform a lookup [22]; when the block size is the same as the memory word, lookups only require a single memory access [23]. This is important in applications in which high-speed operations are required [24]. Other extensions of the Bloom filters have focused on reducing the false positive probability for elements that are frequently checked [25]–[28], or on completely eliminating false positives when the number of elements inserted in the filter is small [29].

To the best of the authors' knowledge, current studies of pollution attacks on Bloom filters have focused only on the original Bloom filter, or on the Counting Bloom filter, i.e., other variants that are widely used, have not been considered. Therefore, it is of interest to study the potential impact of pollution attacks on different types of Bloom filter. This is the focus of this paper that first proposes new pollution attacks on three popular Bloom filter variants: the Block Bloom filter (BBF) [22], the Variable Increment Counting Bloom filter (VI-CBF) [20] and the Fingerprint Counting Bloom filter (FP-CBF) [21]. Their impact is evaluated and a theoretical analysis is provided to show that these filters are significantly more vulnerable to pollution attacks than a traditional Bloom filter. In particular, for the BBF (that is widely used to improve lookup speed), the attacker can increase the false positive probability by several orders of magnitude under some settings. In many systems, this may lead to a significant performance degradation; however, the implementation of these attacks may

978-3-903176-36-2

require a large or in some cases even unfeasible computational effort. Therefore, simplified attack algorithms with a bounded complexity are also proposed. These simplified attacks are also implemented and evaluated to show that false positive rate values close to those of an ideal attack can be achieved in practice. The proposed analytical and experimental results highlight that pollution attacks can have a significant impact for some types of Bloom filters, larger than suggested by previous studies and thus, they need to be considered when designing a system. Similarly, the study of the impact of pollution attacks on other types of Bloom filters is also of interest, so to identify the types that are more vulnerable.

The main contributions of this paper are:

1) To show that the impact of pollution attacks can be completely different depending on the Bloom filter variant.
2) To design attacks for three Bloom filters variants: the Block Bloom filter, the Variable Increment Bloom filter and the Fingerprint Bloom filter.
3) To evaluate the proposed attacks by simulation and show that they are feasible in practical configurations.

The rest of the paper is organized as follows. Section II covers the background on Bloom filters, explaining in some detail the different filter types analyzed (i.e., BBF, VI-CBF and FP-CBF) in this manuscript. Then in section III, existing pollution attacks on traditional Bloom filters are first described, then pollution attacks are presented and theoretically analyzed for the different Bloom filter variants. This section also analyzes the potential increase in false positive probability that may occur in each case as well as the attack complexity; moreover, it describes a generic, yet novel algorithm for the attack. The attacks are evaluated in section IV by considering an adversary with limited computing capacity. The results show that even in this conservative case, the attacker can significantly increase the filter false positive probability. Finally, the paper ends with the conclusion and future work in section V.

II. Preliminaries

A Bloom filter is a data structure that implements approximate membership checking [7], [8]. In many applications, there is a need to check if a given element belongs to some represented sets. For example, network security may require checking whether the source address of a packet is among a blacklist of restricted addresses [9]. This can be done by maintaining the set as a list, or in a hash table and checking it for each element. However, in some cases the table is large and storing it on fast memory is costly or in many cases it is even not feasible. In these scenarios, an approximate membership check rather than an exact one can be used; for example, an approximate check that observes false positives (but not false negatives) can be used to check whether the packet address appears on a blacklist. On a negative indication, the packet can be forwarded safely; however, on a positive indication, the check should be completed with the entire exact representation of the set to determine whether the packet must be blocked, or a false positive has occurred. This approximate check can significantly speed up processing by avoiding the full check for most elements.

TABLE I: Summary of notation

Symbol	Meaning
k	number of hash functions
m	number of bits/counters in the filter
n	number of elements inserted in the filter
b	number of bits per block size in the BBF
L	minimal counter increment in the VI-CBF
f	number of fingerprint bits in the FP-CBF
t	number of elements tested per insertion in the attack algorithm
R_A	ratio of the filter false positive under attack vs. normal operation

In the rest of this section, the original Bloom filter [7] is described first and then, the different filter variants for which pollution attacks are studied in this paper, are also briefly presented. Before going into the details, the notation used in the paper is summarized in Table I.

A. Bloom filters

The structure and operation of a Bloom filter is shown in Figure 1. The filter is formed by an array of m bits that are initially set to zero. Each element x is mapped to k positions using a group of hash functions $h_1(x), \ldots, h_k(x)$. Those positions are set to one when an element is inserted. To check if an element has been stored in the filter, the same positions are checked and when they are all one, a positive is returned. By construction, the Bloom filter cannot have false negatives; instead, false positives can occur as positions $h_i(x)$ may have been set to one when inserting other elements (that happen to map also to some of those positions).

The probability that a lookup (for a randomly selected element x that is not in the set) returns a positive, depends on the number of elements inserted in the filter n, the size of the filter m and the number of hash functions used k. Hence, the false positive probability (FPP) can be approximated by:

$$FPP \approx \left(1 - \left(\frac{m-1}{m}\right)^{n \cdot k}\right)^k. \quad (1)$$

When the size of the filter m is large, a simpler expression can also be used:

$$FPP_{BF} \approx \left(1 - e^{-\frac{k \cdot n}{m}}\right)^k. \quad (2)$$

When m is small (typically less than a few hundred bits) more complex equations are needed to obtain an accurate estimate [30].

The original Bloom filter does not support the removal of elements, because setting bits to zero in the array can generate false negatives for other elements. To support removals, counters instead of bits can be used in the array, such that insertions increment and removals decrement the counters. This variant, known as the Counting Bloom filter (CBF), is one of many arrangements that have been proposed over the years [19]. For example, some other variants have been proposed to reduce the false positive probability of the filters and to make them faster, such that queries can be completed in a single memory access. In the following subsections, three filter variants are briefly described; they are used then to evaluate the effectiveness of pollution attacks.

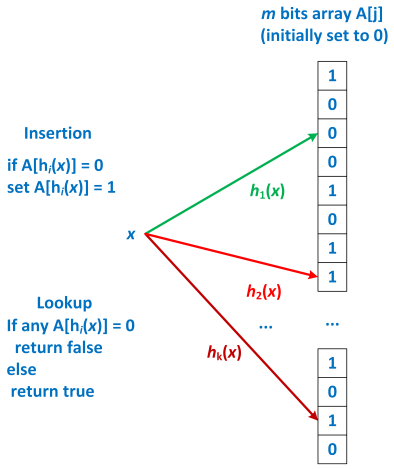

Fig. 1: Illustration of a Bloom filter. Each element is first mapped to k positions using hash functions $h_i(x)$ and those bits are set (insertion) or checked (lookup).

B. Block Bloom filters (BBFs)

A drawback of Bloom filters is the arbitrary location of the k bits mapped by an element. The insertion or lookup of an element might require k memory accesses which can be rather costly. To avoid this issue, the Block Bloom filter (BBF) [22] has been proposed. The structure of a BBF is shown in Figure 2. The filter is divided into blocks of b bits and each element maps to a single block using a hash function $g(x)$. Then, k bits are selected from such a block using a group of k hash functions $h_i(x)$ like in the original Bloom filter. The key advantage of the BBF is that all positions that need to be checked, are in the same block; therefore, if each block has the same size of a memory word, lookups can be completed with just one memory access (the two common configurations are to set the block size to be the same as a memory word, or a cache line, for example 64 or 512 bits [23]).

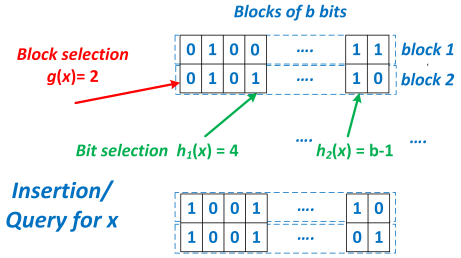

Fig. 2: Illustration of a Block Bloom filter (BBF). Each element is first mapped to a block of b bits using a block selection hash function $g(x)$ and then to k positions in that block using bit selection hash functions $h_1(x), \ldots, h_k(x)$.

The false positive probability of a BBF can be computed from that of a Bloom filter. However, the exact formula of the traditional Bloom filter must be used to obtain accurate results. This is because each block is small [31] and thus the traditional Bloom filter approximations are not accurate. The false positive probability is given by equation 3. In general, the BBF has a larger false positive probability than the traditional Bloom filter [22].

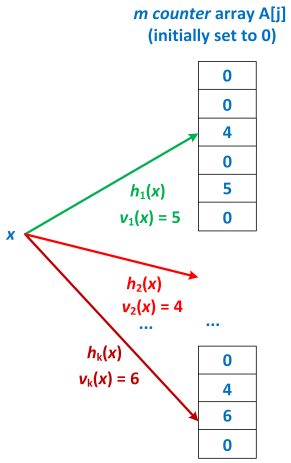

Fig. 3: Illustration of a Variable Increment Counting Bloom filter (VI-CBF) with $L = 4$ and potential increments $D_{L=4} = \{L, L+1, \ldots, 2L-1\} = \{4, 5, 6, 7\}$. For insertion, the increments $v_i(x)$ are added to the counters. For lookups, membership can be eliminated when counters have values lower than $2L$ and do not match the corresponding $v_i(x)$, as for $v_1(x)$ in the example.

C. Variable Increment Counting Bloom filters (VI-CBFs)

The Counting Bloom filter (CBF) uses counters to support removals, but it requires more memory than the original Bloom filter for providing the same false positive probability. Other Bloom filter variants (such as the Variable Increment Counting Bloom filter (VI-CBF)) aim at reducing the false positive probability. The VI-CBF [20] uses variable increments from a set D to increase the counters; for example, $D_L = [L, 2L-1] = \{L, L+1, \ldots, 2L-1\}$. To insert an element, it is mapped to k positions using hash functions $h_i(x)$ and each counter is increased by a variable increment determined by another group of hash functions $v_i(x)$ that select a value from D. To check the membership of an element y, we check whether each counter value mapped by $h_i(y)$ can be comprised of the corresponding variable increment of y. If a given counter has a value in the range $[L, 2L-1]$ and it does not match the increment for element y, then y is not in the set. This significantly reduces the false positive probability when many counters have only one mapped element and thus, they have values in such range. Membership can also be eliminated also in some cases when the counter has a value in the range $[2L, 3L-1]$ depending on the value of the increment for y. The VI-CBF is illustrated in Figure 3.

As indicated in [20], the false positive probability of the VI-CBF can be computed using equation 4 which gives a lower value than for a CBF in many cases, hence making the VI-CBF attractive for practical applications.

D. Fingerprint Counting Bloom filters (FP-CBFs)

Another scheme proposed to reduce the false positive probability of CBFs is the Fingerprint Counting Bloom filter (FP-CBF) [21]. In this scheme, a fingerprint field is stored with each counter, i.e., each position in the filter array stores a counter c_i and a fingerprint fp_i. When adding an element to the filter, its

$$FPP_{Block} \approx \sum_{a=1}^{n} \binom{n}{a} \cdot \left(\frac{1}{m}\right)^{a} \cdot \left(\frac{1}{m}\right)^{n-a} \cdot \left(\frac{b!}{b^{k\cdot(a+1)}} \cdot \sum_{i=1}^{b}\sum_{j=1}^{i} (-1)^{i-j} \frac{j^{k\cdot a} \cdot i^{k}}{(b-i)!j!(i-j)!}\right). \quad (3)$$

$$FFP_{VI} = \left(\left(1-\frac{1}{m}\right)^{nk} + \frac{L-1}{L}\binom{nk}{1}\frac{1}{m}\left(1-\frac{1}{m}\right)^{nk-1} + \frac{(L-1)(L+1)}{6L^2}\binom{nk}{2}\left(\frac{1}{m}\right)^{2}\left(1-\frac{1}{m}\right)^{nk-2}\right)^{k}. \quad (4)$$

$$FFP_{FP} = \left(\left(1-\frac{1}{m}\right)^{nk} + \frac{2^f-1}{2^f}\binom{nk}{1}\frac{1}{m}\left(1-\frac{1}{m}\right)^{nk-1}\right)^{k}. \quad (5)$$

Fig. 4: The false positive probability of the Block Bloom filter (BBF) in equation (3), the Variable Increment Counting Bloom filter (VI-CBF) in equation (4) and the Fingerprint Counting Bloom filter (FP-CBF) in equation (5).

fingerprint $f(x)$ is computed in addition to the k hash functions $h_i(x)$. Then the counters on those positions are incremented and the fingerprints are updated by setting $fp_{h_i(x)}$ as $fp_{h_i(x)}\ xor\ f(x)$. To remove an element, the counters are decremented, and the fingerprint is updated in the same fashion as when adding the element. The use of the fingerprint enables to eliminate the membership when checking x if $c_i = 1$ and $f(x) \neq fp_i$. This is conceptually like the method used in the VI-CBF. The false positive probability of a FP-CBF that uses fingerprints of f bits can be computed by equation 5 and it is slightly lower than that of the VI-CBF for most configurations.

III. Pollution Attacks on Bloom filters

A pollution attack aims at increasing the filter false positive probability by carefully selecting the elements that are inserted [17]; this degrades the performance of the system in which the filter is used. In the original Bloom filter, the false positive probability increases with the number of bits that are set to one; so, the attacker can select elements that do not overlap with the ones previously inserted. This can be easily done when the attacker knows the filter implementation details, including the hash functions $h_i(x)$. It is not straightforward when the attacker can only perform insertions, removals and lookups in the filter, so it sees as a black-box. However, a recent work [18] has shown that an attacker can use the increment or delta observed in the false positive rate when inserting an element to generate pollution attacks.

To measure the effectiveness of the attacks, the attack ratio R_A is utilized; it is defined as the ratio of the false positive probability (FPP) when the filter is polluted versus the FPP of the filter in normal operation:

$$R_A = \frac{FPP_{Attack}}{FPP_{Normal}} \quad (6)$$

This metric quantifies the increase suffered by the filter FPP due to the attack.

In the rest of this section, existing pollution attacks on the original Bloom filter are initially described; then, new pollution attacks for the BBFs, VI-CBFs and FP-CBFs are proposed and theoretically analyzed. The attack complexity and feasibility are then discussed by proposing a generic yet simple attack algorithm.

There are many Bloom filter variants that can be studied. For example, the retouched Bloom filter that reduces false positives by introducing few false negatives [32]. Most of the Bloom filter improvements focus on reducing either the false positive probability or increasing the speed of lookups. In this context, the reasons for choosing BBFs, VICBFs and FP-CBFs is that the BBF is an example of a variant whose objective is increasing the lookup speed even if it implies a small degradation of the false positive probability. The VI-CBF and FP-CBF are examples of variants that try to reduce the false positive probability by adding additional complexity to the filter operations. Therefore, as per the choice of filters we cover both speed and false positive improvements

A. Bloom filters

Pollution attacks on Bloom filters have been proposed by considering both white-box [17] and black-box adversaries [18]; in both scenarios, the attacker tries to ensure that each inserted element sets to one exactly k positions. Therefore, if that goal is achieved, the attacker can construct a filter with n elements that has a false positive probability given by:

$$FPP_{attack} \approx min(\left(\frac{n\cdot k}{m}\right)^{k}, 1). \quad (7)$$

This is achieved by ensuring that each element (for the n inserted elements) sets to one exactly k positions in the filter; the attack ratio is then given by:

$$R_A \approx \left(\frac{\frac{n\cdot k}{m}}{1-e^{-\frac{n\cdot k}{m}}}\right)^{k}. \quad (8)$$

The values of R_A that can be achieved are only moderate as shown next.

B. Block Bloom filters (BBFs)

For BBFs [22], the attacker can be more effective by concentrating the insertions on a few blocks. This can be analyzed by considering the effect of adding an element to a block of b bits that has p bits set. Its false positive probability increases by $(\frac{p+k}{b})^k - (\frac{p}{b})^k$. The value is $(\frac{k}{b})^k$ for the first element $p = 0$, while for the next elements, it can be approximated by:

$(\frac{p^{k-1} \cdot k^2}{b})^k$, i.e., the increment in the false positive probability is larger for larger values of p. Therefore, to maximize the false positive probability, the attacker must concentrate on filling a subset of the blocks only. For example, if $n = 2048$ and $k = 4$ and $b = 64$, the attacker can fill completely $\frac{2048 \cdot 4}{64} = 128$ blocks, such that their false positive probability is one. If the number of blocks is 1024, then the overall false positive probability is approximately $\frac{128}{1024} = \frac{1}{8}$, so it is significantly larger than for the filter during normal operation. In more detail, when inserting n elements, the attacker could completely fill $\lfloor \frac{n \cdot k}{b} \rfloor$ blocks and achieve a false positive probability given by:

$$FPP_{attack} \approx \left(\frac{n \cdot k}{m} \right). \tag{9}$$

Hence, a linear relationship between the number of inserted elements and the false positive probability is found; this is significantly larger than for the traditional Bloom filter and shows the way an attacker can exploit the locality enforced by the BBF to make the attack much more effective.

C. Variable Increment Counting Bloom filters (VI-CBFs)

In the VI-CBFs [20] the contribution of a given position to the false positive probability increases significantly when two or more elements map to it. This occurs because the counter value cannot be used to eliminate the membership in the VI-CBF. For example, consider an element x for which $k-1$ positions return a positive, and membership is eliminated by the k^{th} position that is empty. Then adding one element to that position creates a false positive for x with probability $\frac{1}{L}$ in the VI-CBF. Instead, consider an element y for which $k-1$ positions return a positive, and membership is eliminated by the k^{th} position that stores a single element. Then adding a second element for that position creates a false positive with high probability; the false positive always occurs for the VI-CBF if the counter reaches a value of $3L$. For smaller counter values, the false positive still occurs with high probability. However, if the position already stores two or more elements, adding a new one has little or no impact on the false positive probability. Therefore, to maximize the false positive probability, exactly two elements must map to each position.

For example, for the VI-CBF, when inserting elements and each position gets two elements whose increments add to $3L$, or more, or no element, then the false positive probability is approximately given by:

$$FPP_{attack} \approx \left(\frac{n \cdot k}{2 \cdot m} \right)^k. \tag{10}$$

This corresponds to that of an attack on a traditional Bloom filter in which $n/2$ elements have been inserted.

D. Fingerprint Counting Bloom filters (FP-CBFs)

For the FP-CBF, the same approach used to attack the VI-CBF can be used, and the false positive probability is also given by the equation 10. In the FP-CBF, having two elements mapping to a counter always ensures that the fingerprint cannot be used to reduce the false positive probability.

However, for the FP-CBF, an alternative approach for the attack is to insert only elements with a given fingerprint value, for example $f(x) = 0$, i.e., elements y such that $f(y) \neq 0$ return a negative but elements with $f(y) = 0$ have the false positive probability of an attack on a traditional Bloom filter. Therefore, the overall false positive probability is given by:

$$FPP_{attack} \approx \frac{1}{2^f} \cdot \left(\frac{n \cdot k}{m} \right)^k. \tag{11}$$

Depending on the values of k and f, this attack may be more effective than the one described in the previous subsection. In more detail, it is more effective when $\frac{1}{2^f} > \frac{1}{2^k}$ so when $k > f$. This is evaluated in the next subsection.

E. Attack Potential

To illustrate the differences in the potential of pollution attacks for different types of Bloom filters, the attack ratio R_A has been computed for a filter with $m = 2^{16}$ bits (counters for the VI-CBFs and FP-CBFs) and $k = 2, 4$ for different values of n. For the BBF, the block size b is set to 64 bits and for the VI-CBF, L is set to 4. For the FP-CBF, the number of fingerprint bits f is set to 2 and counters of four bits are assumed in all cases.

The results are summarized in Figures 5 and 6. The first figure shows the false positive rates of the different filters under normal operation, while the second figure shows the attack ratio R_A; so, more sophisticated VI-CBFs and FP-CBFs have the lowest false positive probabilities and the BBF the largest, as also obtained previously in other works found in the technical literature [20], [21], [22].

Consider the potential of pollution attacks; BBFs are significantly more vulnerable, especially at low false positive probabilities. The VI-CBFs and FP-CBFs are also more vulnerable than the traditional Bloom filter. For the FP-CBF, the attack that tries all elements to have the same fingerprint value (labeled as "FP fingerprint" in the figures), is more effective when $k = 4$ is larger. Instead, for $k = 2$, both attacks have the same R_A. Consider the FPP_{attack} expressions for both cases; indeed when $k = 2$ and $f = 2$, they give the same value.

F. Attack Feasibility

In the previous subsection, the potential of pollution attacks has been discussed in terms of the attack ratio R_A that can be attained by the different variants of Bloom filters in the ideal case; however, the attack complexity has not been considered and in some practical cases it may make the attack unfeasible. As an example, consider the attack on the VI-CBF in which n elements (with n as an even integer) are inserted; then when inserting the last element, if the previous insertions have been consistent with the proposed attack, we could have exactly k counters to which only one element maps. The last element must map precisely to these k elements to increment the counters. For a filter with m counters, the probability that an element maps to those counters is given by $\frac{k!}{m^k}$. Taking as an example one of the configurations considered in the previous subsection with $k = 4$ and $m = 2^{16}$, then the probability of finding such an element is lower than $1.3 \cdot 10^{-18}$. Therefore, the time needed to find such an element is not practically feasible. For the BBF, insertion of the last element when it fills completely a block, occurs with a probability given by

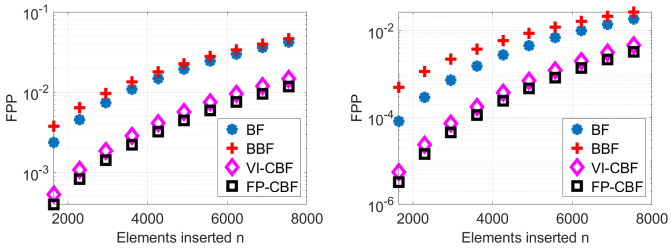

Fig. 5: False positive probability versus number of inserted elements for $k = 2$ (left), and $k = 4$ (right).

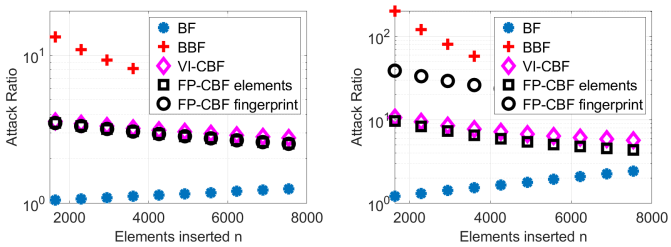

Fig. 6: Attack ratio (R_A) versus number of inserted elements for $k = 2$ (left), and $k = 4$ (right).

$\frac{w}{m} \cdot \frac{k!}{b^k}$. The first term is the probability that an element maps to the block to fill, while the second term is that its k bits map to the k positions that are not one. For the configuration considered in the previous subsection with $k = 4$, $b = 64$ and $m = 2^{16}$, the probability is lower than $1.4 \cdot 10^{-9}$; this is a very reasonable value, but it still needs significant computing time to find such an element. As an example, the proposed attack has been run on a BBF with the above settings assuming that the hash functions are known. The number of elements tested to build the attack set are shown in Figure 7; this figure gives the average over four runs of the attack. The values are only slightly larger than those estimated for the last element that according to the formula, are approximately 715 million. This is also intuitive because the process of finding elements for insertion can be easily accomplished at the beginning, but it becomes increasingly difficult as more elements are inserted and the target blocks are almost filled. Therefore, the expected number of elements to get one valid for the last insertion seems to be an initial estimate of the complexity needed to build the attack set.

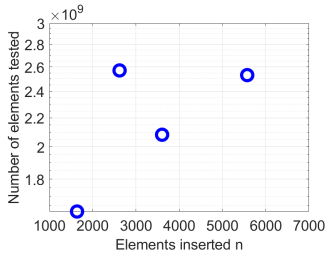

Fig. 7: Number of elements tested to build the attack set for a Block Bloom filter (BBF) with $k = 4$, $b = 64$ and $m = 2^{16}$. The reported values are averages over four runs.

G. Attack Algorithm

The discussion of the previous subsection shows that the implementation of the attack to find the theoretical attack ratio R_A may require to test a large and even unfeasible number of elements depending on the filter type and configuration. However, in many scenarios, it is sufficient to obtain a significantly large value of the attack ratio for an attacker; therefore, it is of interest to consider the values of R_A that can be achieved with a reasonable computational complexity. Similarly, in some cases, the attacker may not have access to the implementation details of the filters and can only observe the effects (in terms of false positive probability) of the insertion [18].

Consider a general algorithm that can be used for both black-box and white-box attacks and with a bounded computational complexity. The algorithm proposed in this paper is similar to [18] for black-box attacks; when inserting an element, t random candidates are generated and evaluated, and the best candidate is inserted. The evaluation is based on a parameter Δ that depends on the type of attack and filter. For a black-box attack, Δ is based on the increment in the false positive probability induced by inserting an element. This can be estimated by measuring the false positive probability, inserting the candidate element, measuring the false positive probability again, computing the difference to obtain Δ and removing the element tested as discussed in [18]. For a white-box attack, the elements can be selected based on their desired features for each attack algorithm, such as examples, for the BBF, such as those that fall on the blocks selected to be filled and that set

Algorithm 1 Insertion of an element on attack

```
Randomly generate a set of test elements T with t elements
Initialize Δ_max to zero
for x in T do
    Evaluate Δ for x
    if Δ > Δ_max then
        z = x;
        Δ_max = Δ
    end if
end for
insert z;
```

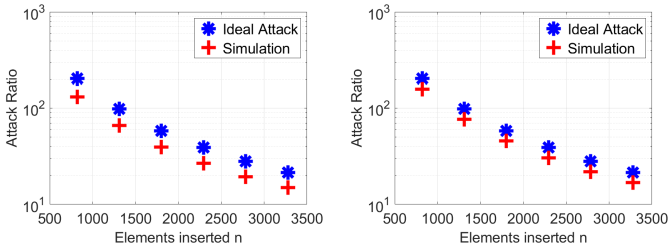

Fig. 8: Block Bloom filter (BBF): Attack ratio for $b = 64$ and $m = 2^{15}$ when $t = 1000$ (left) and $t = 10^5$ (right).

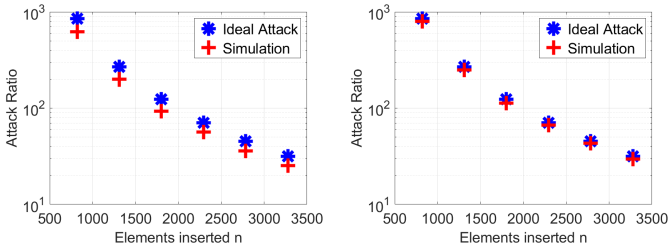

Fig. 9: Block Bloom filter (BBF): Attack ratio for $b = 512$ and $m = 2^{15}$ when $t = 1000$ (left) and $t = 10^5$ (right).

more bits to one in those blocks. In more detail, the criteria for selection among the t candidates in Algorithm 1 for each type of filters are given as follows:

1) BBF: Δ is not zero only for elements that map to the target blocks to be filled; among them the value of Δ is given by the number of bits set to one when inserting the element.
2) VI-CBF: Δ is the number of positions that store a single element to which the element maps.
3) FP-CBF (elements attack): Δ is the number of positions that store a single element to which the element maps.
4) FP-CBF (fingerprint attack): Δ is not zero only for elements with the target value of $f(x)$ among them the value of Δ is given by the number counters set to one by the element.

The algorithm to insert each element in the attack set is given as Algorithm 1. The elements in T are evaluated and the element in T with the largest value is inserted. The proposed algorithm is independent of the filter implementation and has a bounded number of elements tested, given by $n \cdot t$.

IV. Evaluation

As discussed at the end of the previous section, to evaluate the potential of pollution attacks in practical scenarios, a generic algorithm with bounded complexity is considered. In the experiments, a white-box attack is simulated and the criteria to select among the t candidates in Algorithm 1 are those described in the previous section. The following settings have been used in the experiments:

- Filter size (in bits/counters): $m = 2^{15}$.
- Number of hash functions: $k = 4$.
- Block size for the BBF: $b = 64, 512$.
- Increments for the VI-CBF: $L = 4, 8$.
- Number of fingerprint bits for the FP-CBF: $f = 2, 3$.

In all experiments, different values for the number of elements n are evaluated; the values of the tested n elements are in the range $\frac{m}{40}$ to $\frac{m}{10}$ which correspond to 40 to 10 bits per element.

The experiments are intended to show that large values of the attack ratio R_A (in some cases close to the ideal ones) can be achieved with limited computational complexity; therefore, experiments are run for each of the Bloom filter variants considered in this paper, the results are then compared with those found by the theoretical analysis presented in the previous section.

In the first experiment, we assess the effectiveness of the algorithm for the BBF; two block sizes $b = 64, 512$ were simulated with $t = 1000, 10^5$ and the R_A is measured and compared to the theoretical value. The results are summarized in Figure 8 for $b = 64$ and in Figure 9 for $b = 512$; when $t = 10^5$, the ratios are close to the ideal values, however for $t = 1000$, the ratios are lower, but still above 10x in all cases. Therefore, an attacker can significantly increase the false positive probability in both cases. Considering the block size b, the attack is more effective when $b = 512$, because in such case, the false positive of the filter under normal operation is lower and this makes R_A larger. Finally, these experimental results confirm that BBFs are very vulnerable to pollution attack with increases in the false positive probability that approaches a factor of 1000 for large blocks and number of bits per element.

In the second experiment, the attack is simulated for the VI-CBF when $L = 4, 8$; again, two values of $t = 1000, 10^5$ are evaluated and the R_A is measured and compared to the theoretical value. The results are shown in Figures 10 and 11; when $t = 1000$, the simulated attack ratios are lower than the ideal ones. When $t = 10^5$, the values are larger, but still far from the ideal ones; this occurs because finding ideal candidates for insertion is difficult as they need to map to positions that store a single element (as discussed previously).

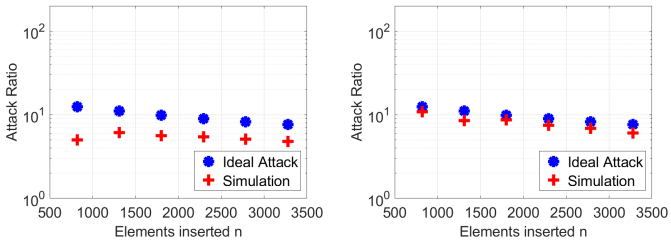

Fig. 10: Variable Increment Counting Bloom filter (VI-CBF): Attack ratio for $L = 4$ and $m = 2^{15}$ when $t = 1000$ (left) and $t = 10^5$ (right).

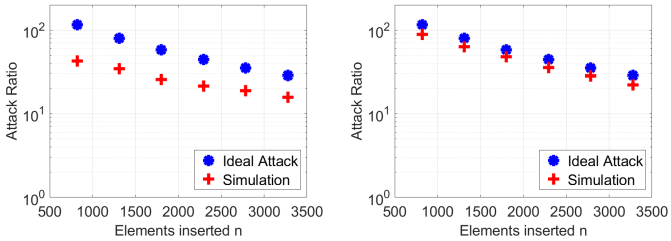

Fig. 11: Variable Increment Counting Bloom filter (VI-CBF): Attack ratio for $L = 8$, $m = 2^{15}$ when $t = 1000$ (left) and $t = 10^5$ (right).

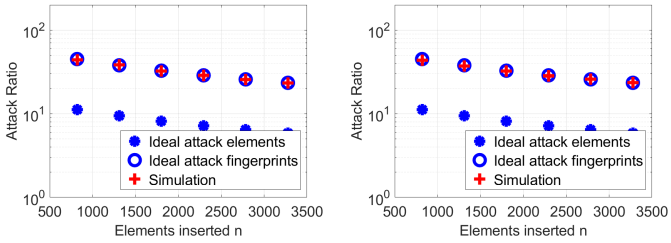

Fig. 12: Fingerprint Counting Bloom filter (FP-CBF): Attack ratio for $f = 2$ and $m = 2^{15}$ when $t = 1000$ (left) and $t = 10^5$ (right).

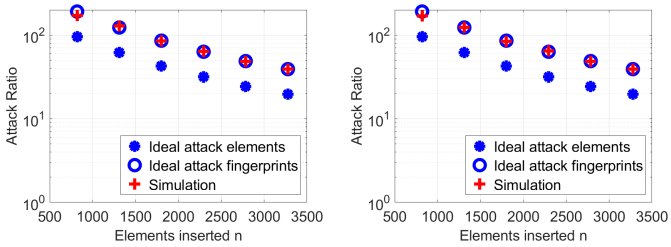

Fig. 13: Fingerprint Counting Bloom filter (FP-CBF): Attack ratio for $f = 3$ and $m = 2^{15}$ when $t = 1000$ (left) and $t = 10^5$ (right).

When considering the minimum increment L, the attack is more effective for $L = 8$ than for $L = 4$ (as expected), because increments are more powerful to reduce the false positive probability under normal operation (and the attack removes that reduction). These results confirm that the VI-CBF is also more vulnerable to pollution attacks than the traditional Bloom filter, but less than the BBF.

In a third experiment, the fingerprint attack is run on a FP-CBF with $f = 2, 3$ bit fingerprints. The fingerprint attack is selected, because the theoretical analysis has shown that it is more effective than the elements attack. The results are presented in Figures 12 and 13; in this case, the attack achieves the ideal values of R_A for both $t = 1000$ and $t = 10^5$. This occurs because only elements with a given fingerprint value must be found, while avoiding collisions. The attack is more effective as f increases, because the benefit of using the fingerprints is significant and the attack basically reduces that benefit. Finally, FP-CBFs are also quite vulnerable to pollution attacks, significantly more than the traditional Bloom filters but less than the BBFs.

In summary, these simulation experiments show that an attacker can significantly increase the false positive probability for all considered Bloom filter types and achieve values of R_A close to the ideal ones. The BBF is the most vulnerable type followed by the FP-CBF and the VI-CBF. Finally, comparing with the traditional Bloom filter, all three Bloom filter types are significantly more vulnerable.

V. Conclusion and Future Work

In this paper, pollution attacks for several widely used types of Bloom filters have been studied; this has led to

new pollution attacks for the Block Bloom filters (BBFs), the Variable Increment Counting Bloom filters (VI-CBFs) and the Fingerprint Counting Bloom filters (FP-CBFs). A theoretical analysis has shown that the proposed attacks are able to increase the false positive probability significantly more than existing attacks on the original Bloom filter. This has been corroborated by implementing and simulating the attacks to show that the false positive probability can be increased by at least an order of magnitude in many settings. Therefore, these types of Bloom filters are more vulnerable to pollution attacks that can severely degrade their performance. In particular, the BBF is the most vulnerable type with increases in the false positive rate close to 1000x for some settings, followed by the FP-CBF that exceeds 100x for some configurations and the VI-CBF that is close to 100x when the number of bits per element is large. This compares with smaller increases in the traditional Bloom filter that are below 5x for all the configurations considered.

After showing that some types of Bloom filters are particularly vulnerable to pollution attacks, this work is being extended to design and implement schemes to detect and mitigate the attacks. For the BBFs, the attack can be detected by monitoring the difference in the number of ones between the block with more ones and the block with fewer ones; if this difference is too large, it is an indication of an attack. Similarly, for the VI-CBFs and FP-CBFs, the number of counters with only one element and the ones with more two elements can be monitored; when most elements have two elements and the total number of elements is not large, it is likely that the filter is suffering a pollution attack.

More generally, the analysis of the expected distribution of the filter contents theoretically enables the definition of thresholds to detect anomalies caused by an attack. The design and evaluation of these attack detection techniques are left for future work, as well as further research to propose other potentially more efficient attack detection techniques.

ACKNOWLEDGMENT

Pedro Reviriego was supported by the ACHILLES project PID2019-104207RB-I00, the Go2Edge network RED2018-102585-T and project TAPIR-CM P2018/TCS-4496.

REFERENCES

[1] S. A. Crosby and D. S. Wallach, "Denial of service via algorithmic complexity attacks," in *USENIX Security Symposium*, 2003.

[2] L. Csikor, D. M. Divakaran, M. S. Kang, A. Kőrösi, B. Sonkoly, D. Haja, D. P. Pezaros, S. Schmid, and G. Rétvári, "Tuple space explosion: A denial-of-service attack against a software packet classifier," in *ACM International Conference on Emerging Networking Experiments And Technologies (CoNEXT)*, 2019.

[3] G. Cormode, "Data sketching," *Commun. ACM*, vol. 60, no. 9, p. 48–55, 2017.

[4] D. Clayton, C. Patton, and T. Shrimpton, "Probabilistic data structures in adversarial environments," in *ACM Conference on Computer and Communications Security (SIGSAC)*, 2019.

[5] I. Mironov, M. Naor, and G. Segev, "Sketching in adversarial environments," *SIAM J. Comput.*, vol. 40, no. 6, pp. 1845–1870, 2011.

[6] M. Hardt and D. P. Woodruff, "How robust are linear sketches to adaptive inputs?" in *ACM Symposium on Theory of Computing Conference (STOC)*, 2013.

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

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

[9] P. Reviriego, J. Martínez, D. Larrabeiti, and S. Pontarelli, "Cuckoo filters and Bloom filters: Comparison and application to packet classification," *IEEE Transactions on Network and Service Management (TNSM)*, 2021.

[10] K. Li and Z. Zhong, "Fast statistical spam filter by approximate classifications," in *ACM SIGMETRICS/Performance*, 2006.

[11] J. Yan and P. L. Cho, "Enhancing collaborative spam detection with Bloom filters," in *IEEE Annual Computer Security Applications Conference (ACSAC)*, 2006.

[12] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of Bloom filters for distributed systems," *IEEE Communications Surveys Tutorials*, vol. 14, no. 1, pp. 131–155, 2012.

[13] T. Wang, W. Zhu, Q. Ma, Z. Shen, and Z. Shao, "ABACUS: Address-partitioned Bloom filter on address checking for uniqueness in IoT blockchain," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020.

[14] O. Rottenstreich, "Sketches for blockchains," in *International Conference on Communication Systems and Networks (COMSNETS)*, 2021.

[15] O. Rottenstreich and I. Keslassy, "The Bloom paradox: When not to use a Bloom filter," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 703–716, 2015.

[16] M. Naor and Y. Eylon, "Bloom filters in adversarial environments," *ACM Trans. Algorithms*, vol. 15, no. 3, 2019.

[17] T. Gerbet, A. Kumar, and C. Lauradoux, "The power of evil choices in Bloom filters," in *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015.

[18] P. Reviriego and O. Rottenstreich, "Pollution attacks on counting Bloom filters for black box adversaries," in *International Conference on Network and Service Management (CNSM)*, 2020.

[19] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, "Optimizing Bloom filter: Challenges, solutions, and comparisons," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1912–1949, 2019.

[20] O. Rottenstreich, Y. Kanizo, and I. Keslassy, "The variable-increment counting Bloom filter," *IEEE/ACM Transactions on Networking*, vol. 22, no. 4, pp. 1092–1105, 2014.

[21] S. Pontarelli, P. Reviriego, and J. A. Maestro, "Improving counting Bloom filter performance with fingerprints," *Information Processing Letters*, vol. 116, no. 4, pp. 304–309, 2016.

[22] U. Manber and S. Wu, "An algorithm for approximate membership checking with application to password security," *Inf. Process. Lett.*, vol. 50, no. 4, p. 191–197, 1994.

[23] Y. Qiao, T. Li, and S. Chen, "Fast Bloom filters and their generalization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 93–103, 2014.

[24] H. Lang, T. Neumann, A. Kemper, and P. Boncz, "Performance-optimal filtering: Bloom overtakes cuckoo at high throughput," *VLDB Endow.*, vol. 12, no. 5, p. 502–515, 2019.

[25] J. Bruck, J. Gao, and A. Jiang, "Weighted Bloom filter," in *IEEE International Symposium on Information Theory (ISIT)*, 2006.

[26] M. A. Bender, M. Farach-Colton, M. Goswami, R. Johnson, S. McCauley, and S. Singh, "Bloom filters, adaptivity, and the dictionary problem," in *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 2018.

[27] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *ACM SIGMOD*, 2018.

[28] M. Mitzenmacher, "A model for learned Bloom filters and optimizing by sandwiching," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[29] S. Z. Kiss, Hosszu, J. Tapolcai, L. Rónyai, and O. Rottenstreich, "Bloom filter with a false positive free zone," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 18, no. 2, pp. 2334–2349, 2021.

[30] K. Christensen, A. Roginsky, and M. Jimeno, "A new analysis of the false positive rate of a Bloom filter," *Information Processing Letters*, vol. 110, no. 21, pp. 944 – 949, 2010.

[31] P. Reviriego, K. Christensen, and J. A. Maestro, "A comment on "Fast Bloom filters and their generalization"," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 303–304, 2016.

[32] B. Donnet, B. Baynat, and T. Friedman, "Retouched bloom filters: Allowing networked applications to trade off selected false positives against false negatives," in *ACM International Conference on Emerging Networking Experiments And Technologies (CoNEXT)*, 2006.