Adversary Resilient Learned Bloom Filters

Allison Bishop[1,2][0000−0003−3986−8985] and Hayder Tirmazi[1][0009−0008−9360−9662]

[1] City College of New York
[2] Proof Trading
abishop@ccny.cuny.edu, stirmaz000@citymail.cuny.edu

Abstract. Creating an adversary resilient Learned Bloom filter [1] with provable guarantees is an open problem [2]. We define a strong adversarial model for the Learned Bloom Filter. We also construct two adversary resilient variants of the Learned Bloom Filter called the Uptown Bodega Filter and the Downtown Bodega Filter. Our adversarial model extends an existing adversarial model designed for the classical (i.e not "learned") Bloom Filter by Naor and Yogev [3] and considers computationally bounded adversaries that run in probabilistic polynomial time (PPT). We show that if pseudo-random permutations exist, then a secure Learned Bloom Filter may be constructed with λ extra bits of memory and at most one extra pseudo-random permutation in the critical path. We further show that, if pseudo-random permutations exist, then a *high utility* Learned Bloom Filter may be constructed with 2λ extra bits of memory and at most one extra pseudo-random permutation in the critical path. Finally, we construct a hybrid adversarial model for the case where a fraction of the workload is chosen by an adversary. We show realistic scenarios where using the Downtown Bodega Filter gives better performance guarantees compared to alternative approaches in this model.

Keywords: Pseudorandom permutations · Adversarial Artificial Intelligence · Probabilistic Data Structures.

1 Introduction

Bloom filters are probabilistic data structures that solve the Approximate Membership Query Problem. A Bloom filter representing a set S may have false positives ($s \notin S$ may return true) but does not have false negatives ($s \in S$ is always true). The data structure now known as a "Bloom" filter was initially proposed as method 2 in the section "Two Hash-Coding Methods with Allowable Errors" in a 1970 paper by Burton H. Bloom [4][5]. Bloom filters are used in databases, cryptography, computer networking, social networking [6] and network security [7]. Figure 1 provides a helpful illustration of a traditional Bloom filter and the insert and check operations we introduce below.

Definition 1 (Bloom filter). *A Bloom filter for representing set S with cardinality n is a zero initialized array of m bits. A Bloom filter requires k independent hash functions h_i such that the range of each h_i is the set of integers $\{1, \ldots, m\}$ [7].*

Most mathematical treatments such as Mitzenmacher and Broder [7] make the convenient assumption that each h_i maps each item in the universe to a random number uniformly over the (integer) range $[1, m]$.

Operation 1 (Insert). *For each element $x \in S$, the bits $h_i(x)$ are set to 1 for $i \in [1, k]$.*

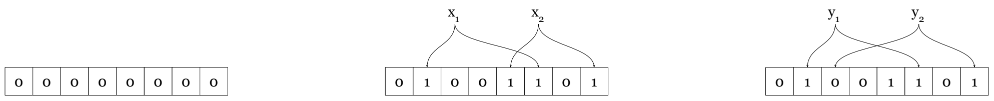

Fig. 1: Example of a Bloom filter with $m = 8$ and $k = 2$. Initially, all m bits are unset. Each element x_i is hashed k times, and each corresponding bit is set. To check each element y_i, the element is hashed k times. If any corresponding bit is unset, the element y_i is not in set S (with probability 1). If all corresponding bits are set, the element y_i is either in set S or the element y_i has caused the Bloom filter to return a false positive

If a bit already set to 1 is set to 1 again, its value remains 1 i.e a double set does not flip the bit back to 0.

Operation 2 (Check)**.** *For an element x, we return true if all $h_i(x)$ map to bits that are set to* 1*. If there exists an $h_i(x)$ that maps to a bit that is* 0*, we return false.*

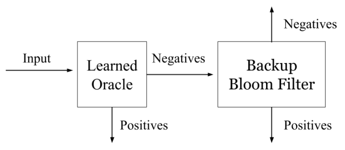

Fig. 2: Example of a learned Bloom filter with a learned Oracle and a backup Bloom filter that only checks values that are with high probability negative in the learned Oracle, to ensure a one-sided error bound (i.e only false positives and no false negatives).

The *Learned* Bloom filter is a novel data structure proposed by Kraska et al [1] in 2018. A mathematical model and guarantees for the Learned Bloom filter are provided in Mitzenmacher et al [8]. Learned Bloom filters use a learned model for the set being represented. They provide better performance on the false positive rate while maintaining the guarantee of having no false negatives. We show an example of a Learned Bloom filter adapted from Mitzenmacher et al [8] in figure 2.

2 Motivation

The Bloom Filter and its variants have numerous applications in computing [9] [7]. We borrow discussion on Bloom Filter applications from the survey by Tarkoma et al [9]. The Bloom Filter may be implemented in kernel space in a Linux network driver for performant filtering of network packets. Loop detection in network protocols, and multicast forwarding engines may also utilize the Bloom Filter. Deep Packet Scanners and Packet Classifiers have also found the Bloom Filter helpful for improving efficiency. Bloom Filters may be used to detect heavy flows in network traffic from the vantage point of a router. The Bloom Filter has also been used in the OPUS system [9] that stores a list of words that involve poor password choices encouraging users to select better passwords. The Bloom Filter has also found success in the detection of hash tampering in network attached disks. Google's BigTable system uses the Bloom Filter to minimize disk reads. Apache Hadoop also uses the Bloom Filter as an optimization in the reduce stage of its map/reduce implementation. Other applications of the Bloom Filter include uses in the realms of peer-to-peer networking, and caching.

2.1 The Need for Security

A large number of the applications of The Bloom Filter involve critical infrastructure [10]. It is possible to forge false positives in a naively implemented Bloom Filter [10] allowing an adversary to make the Bloom Filter deviate from its behavior. Gerbet et al [10] show practical attacks on the Scrapy web-spider, the Bitly Dablooms spam filter, and the Squid web cache. Naor and Yogev [3] motivate the need for securing the Bloom Filter by considering a white-list of email addresses for the purposes of spam filtering. In their scenario, an adversary that can forge false positives may easily infiltrate the spam filter.

3 Related Work

Section 3.1 provides a thorough overview of prior work on adversarial models and security of the classical Bloom Filter. Similarly, Section 3.2 discusses prior work on the security of the Learned Bloom Filter.

3.1 Classical Bloom Filter

Gerbet et al[10] suggest practical attacks on traditional bloom filters and the use of universal hash functions and message authentication codes (MACs) to mitigate a subset of those attacks. Naor and Yogev [3] define an adversarial model for the classical Bloom Filter and use it to prove that (1) for computationally bounded adversaries, non-trivial adversary resilient Bloom filters exist $\iff$ one-way functions exist, and (2) for computationally unbounded adversaries, there exists a Bloom filter that is secure against t queries while using only $\mathcal{O}(n \log \frac{1}{\epsilon} + t)$ bits of memory. n is the size of the set and ϵ is the desired error. We borrow their idea of using Pseudorandom Permutations for the classical Bloom Filter and apply it to the Learned Bloom Filter.

Clayton et al [11] analyze not only the classical Bloom Filter, but also the Counting Bloom Filter, and the Count-Min Sketch, in an adversarial setting. Clayton et al use a stronger adversarial model than Naor and Yogev [3], allowing an adversary to perform insertions and giving an adversary access to the internal state of the classical Bloom Filter. Clayton et al propose the use of salts and keyed pseudorandom functions for securing the classical Bloom Filter. They do not address Learned Probabilistic Data Structures including the Learned Bloom Filter. Both Naor and Yogev, and Clayton et al, perform their analysis in a game-based setting.

Filic et al [12] investigate the adversarial correctness and privacy of the classical Bloom Filter and an insertion-only variant of the Cuckoo Filter. They use a stronger adversarial model than

Naor and Yogev [3] allowing an adversary to insert entries into the Bloom Filter and query for the internal state of the Bloom Filter. Unlike our work, Filic et al [12] do not address the adversarial correctness of the Learned Bloom Filter or its variants. Filic et al [12] perform their analysis in a simulator-based setting.

3.2 Learned Bloom Filter

The authors are only aware of one prior work that addresses the Learned Bloom Filter in an adversarial setting, Reviriego et al [2]. They propose a practical attack on the Learned Bloom Filter. They suggest two possible mitigations for their proposed attack: swapping to a classical Bloom Filter upon detection of the attack, or adding a second backup Bloom Filter. However, they do not provide any provable guarantees on the performance of the Learned Bloom Filter in the presence of adversaries. They leave the security of the Learned Bloom Filter as an open problem in their work.

4 Adversarial Model

We highlighted in the Section 1 our proposed outcomes for the problem of securing Learned Bloom Filters. To reiterate, we proposed (1) defining a strong adversarial model for Learned Bloom filters, (2) proving guarantees on the performance of Learned Bloom filters under the adversarial model, (3) exploring new vulnerabilities in Learned Bloom filters, and (4) constructing an adversary resilient Learned Bloom filter. We consider proposed solutions for outcomes 3 and 4 respectively. To build the tools necessary for analysing such proposed solutions, we first make some progress towards outcome 1.

We describe the adversarial model of [3] and discuss how it can be extended to create an adversarial model for Learned Bloom Filters. We refer to the adversarial model defined by Naor and Yogev [3] as the *classical* adversarial model. Section 4.1 contains a treatment of the classical adversarial model. We also introduce a stronger adversary than the one described in Naor and Yogev's [3] model that has access to the internal state of the Bloom Filter. Section 4.2 introduces a definition of the Learned Bloom Filter adapted from [8] and discusses extensions to the classical adversarial model to make it work with Learned Bloom Filters.

4.1 Classical Adversarial Model

Let S be a finite set of cardinality n in a suitable finite universe U of cardinality u. Let M be a compressed representation of S. Let r be any random string and M_r^S be a compressed representation of S with r. Let λ be a security parameter. Let $A = (A_C, A_Q)$ be any propabilistic polynomial time (PPT) adversary.

Definition 2. *We define C to be a setup algorithm such that $C(1^\lambda, S) = M$. C may be randomized. We define C_r, the randomized version of C, to be a setup algorithm such that $C_r(1^\lambda, S) = M_r^S$.*

As a running example, consider the set $S = x$ in a classical Bloom Filter that uses 2 hash functions, h_1, h_2 and 4 bits such that $h_1(x) = 1$ and $h_2(x) = 3$. A trivial deterministic setup algorithm, on input S, would then generate the representation $M_r^S = 1010$.

Definition 3. *We define Q_s to be a query algorithm such that, given element x, $Q(M, x) = 1$ if $x \in S$, and $Q(M, x) \in \{0, 1\}$ if $x \notin S$. Q_s may not be randomized and may not change M.*

In our running example, a trivial query algorithm returns 1 if and only if all hashes for an element return indexes that are set. Consider a new element y for which $h_1(y) = 1$ and $h_2(y) = 2$. With $M_r^S = 1010$, $Q(M_r^S, x)$ returns 1 since both indices 1 and 3 are set, however $Q(M_r^S, y)$ return 0 as index 2 is not set.

Definition 4. *We define Q_u to be a query algorithm similar to Q_s differing only in that Q_u may be randomized and it may change M after each query.*

We now give a precise definition for a classical Bloom Filter in an adversarial setting.

Definition 5. *Let a Bloom Filter be a data structure $B = (C_r, Q)$ where C_r obeys Definition 2 and Q obeys either Definition 3 or Definition 4.*

We define a special class of Bloom Filters which were coined "steady" Bloom Filters by Naor and Yogev [3]. Steady Bloom Filters do not change their internal representation M_r^S after the setup algorithm C_r has executed. In other words, only query algorithms of the type Q_s are permitted in the steady setting and query algorithms of the type Q_u are not permitted.

Definition 6. *Let a steady (n, ϵ)-Bloom Filter be a Bloom Filter $B_s = (C_r, Q_s)$ such that Q_s obeys Definition 3 and $\forall x \in U$, it holds that*

1. *Completeness: $\forall x \in S : P[Q_s(C_r(S), x) = 1] = 1$*
2. *Soundness: $\forall x \notin S : P[Q_s(C_r(S), x) = 1] \leq \epsilon$*

where the probabilities are taken over C_r.

Now we construct our first adversarial challenge for the classical Bloom Filter in the steady setting. A probabilistic polynomial time (PPT) adversary A, as defined at the start of this section, is given a security parameter $1^{\lambda + n \log(u)}$ and is allowed to construct a set S. The set S is then given to construction algorithm C_r along with the security parameter to yield representation M_r^S. The adversary is allowed t queries to the query algorithm Q_s for which it is provided the results. After the t queries, the adversary must output an element x^*. If x^* is a false positive, the adversary wins the challenge. Otherwise, the adversary is loses the challenge. We define this precisely in Challenge 1

Challenge 1. *We denote this challenge as ${\Lambda^1}_{A,t}(\lambda)$.*

1. $S \leftarrow A_C(1^{\lambda + n \log(u)})$
2. $M_r^S \leftarrow C_r(1^{\lambda + n \log(u)}, S)$
3. $x^* \leftarrow A_Q^{Q_s(M_r^S, \cdot)}(1^{\lambda + n \log(u)}, S)$. *$A_Q$ performs at most t queries $x_1, \ldots, x_t$ to $Q(M_r^S, \cdot)$.*
4. *If $x^* \notin S \cup \{x_1, \ldots, x_t\}$ and $Q_s(M_r^S, x^*) = 1$, output 1. Otherwise, output 0.*

We now define an adversarial resilient classical Bloom Filter based on the random variable ${\Lambda^1}_{A,t}(\lambda)$.

Definition 7. *Let an (n, t, ϵ)-adversarial resilient steady Bloom Filter be any steady Bloom Filter for which it holds that, $\forall \lambda > n \in \mathbb{N}, P[\Lambda^1_{A,t}(\lambda) = 1] \leq \epsilon$.*

Now, we create an extension to Naor and Yogev's [3] model, introducing a stronger adversary that has access to the internal state of the Bloom Filter.

Definition 8. *We define R to be an oracle that returns the compressed representation of S with r, M_r^S, at time t*

Now we construct our second adversarial challenge for the classical Bloom Filter in the steady setting. This challenge is almost identical to the first challenge with the only different being the adversary is allowed access to oracle R. We define our second challenge more precisely in Challenge 2.

Challenge 2. *We denote this challenge as ${\Lambda^2}_{A,t}(\lambda)$.*

1. $S \leftarrow A_C(1^{\lambda+n\log(u)})$
2. $M_r^S \leftarrow C_r(1^{\lambda+n\log(u)}, S)$
3. A_Q *is allowed access to oracle* R
4. $x^* \leftarrow A_Q^{Q_s(M_r^S,\cdot)}(1^{\lambda+n\log(u)}, S)$. A_Q *performs at most* t *queries* $x_1, \ldots, x_t$ *to* $Q(M_r^S, \cdot)$.
5. *If* $x^* \notin S \cup \{x_1, \ldots, x_t\}$ *and* $Q_s(M_r^S, x^*) = 1$, *output* 1. *Otherwise, output* 0.

Analogous to Definition 7, we now define an adversarial reveal resilient classical Bloom Filter based on the random variable $\Lambda^2_{A,t}(\lambda)$.

Definition 9. *Let an (n, t, ϵ)-adversarial* ***reveal*** *resilient steady Bloom Filter be any steady Bloom Filter for which it holds that, $\forall \lambda > n \in \mathbb{N}, P[\Lambda^2_{A,t}(\lambda) = 1] \leq \epsilon$.*

4.2 Learned Adversarial Model

In this section, we provide a mathematical model for the Learned Bloom Filter adapted from [8]. Consider a set of elements $\mathcal{K} \subset S$ and a set of elements $\mathcal{U}$ such that $\forall u \in \mathcal{U}, u \notin S$. We form a dataset $\mathcal{D} = \{(x_i, y_i = 1) | x_i \in \mathcal{K}\} \cup \{(x_i, y_i = 0) | x_i \in \mathcal{U}\}$.

Definition 10. *Let a Bloom Filter Learning Model, $l : U \mapsto [0, 1]$, be any function that maps elements in a suitable finite universe to a probability.*

Let L_r be any construction algorithm for l. We train a Bloom Filter Learning Model, l, by running algorithm L_r on $\mathcal{D}$. Let $l(x)$ be the probability estimate from the learning model that x is an element in S. A value τ may be chosen as a threshold. When $l(x) \geq \tau$ then the Learned Bloom Filter considers x to be an element of S. Otherwise, the Learned Bloom Filter passes x on to the classical Backup Bloom Filter. Figure 2 provides a helpful illustration.

Definition 11. *Let a Learned Bloom Filter, $\tilde{B} = (l, \tau, C_r, Q)$ where l obeys Definition 10, τ is a given threshold for l, C_r obeys Definition 2, and Q obeys either Definition 3 or Definition 4. (C_r, Q) form a classical Bloom Filter (Definition 5), which we refer to as a Backup Bloom Filter. The Backup Bloom Filter holds the set $\{x : x \in S | l(x) < \tau\}$.*

It is trivial to define a steady Learned Bloom Filter with completeness and soundness properties analogous to the ones outlined in Definition 6 for a classical Bloom Filter.

Now, we construct an adversarial model for the Learned Bloom Filter. We assume our adversary A has oracle access to the Bloom Filter Learning Model l, the construction algorithm C_r, and the query algorithm Q.

Challenge 3. *We denote this challenge as ${\Lambda^{1l}}_{A,t}(\lambda)$.*

1. $S, \mathcal{D} \leftarrow A_C(1^{\lambda+n\log(u)})$
2. $M_r^S \leftarrow C_r(1^{\lambda+n\log(u)}, S)$
3. $l \leftarrow L_r(1^{\lambda+n\log(u)}, \mathcal{D})$
4. $x^* \leftarrow A_Q^{Q_s(l, M_r^S, \cdot)}(1^{\lambda+n\log(u)}, S)$. *$A_Q$ performs at most t queries $x_1, \ldots, x_t$ to $Q(M_r^S, \cdot)$.*
5. *If $x^* \notin S \cup \{x_1, \ldots, x_t\}$ and $Q_s(M_r^S, x^*) = 1$, output 1. Otherwise, output 0.*

Note that the adversary may not choose threshold τ. If the adversary is allowed to choose threshold τ, then Challenge 3 is easily succeeded by choosing $\tau = 0$. We are now ready to formally define security for steady Learned Bloom Filters.

Definition 12. *Let an (n, t, ϵ)-adversarial resilient steady Learned Bloom Filter be any steady Learned Bloom Filter for which it holds that, $\forall \lambda > n \in \mathbb{N}, P[\Lambda^{1l}_{A,t}(\lambda) = 1] \leq \epsilon$.*

We now propose a stronger adversary that has access to the internal state of the Learned Bloom Filter. Recall the definition of oracle R (Definition 8). We define an analogous oracle for the learning model.

Definition 13. *We define LR to be an oracle that returns the internal state of the learning model l trained on dataset $\mathcal{D}$ with r*

Challenge 4. *We denote this challenge as ${\Lambda^{2l}}_{A,t}(\lambda)$.*

1. $S, \mathcal{D} \leftarrow A_C(1^{\lambda+n\log(u)})$
2. $M_r^S \leftarrow C_r(1^{\lambda+n\log(u)}, S)$
3. $l \leftarrow L_r(1^{\lambda+n\log(u)}, \mathcal{D})$
4. *A_Q is allowed access to oracles R and LR*
5. $x^* \leftarrow A_Q^{Q_s(l, M_r^S, \cdot)}(1^{\lambda+n\log(u)}, S)$. *$A_Q$ performs at most t queries $x_1, \ldots, x_t$ to $Q(M_r^S, \cdot)$.*
6. *If $x^* \notin S \cup \{x_1, \ldots, x_t\}$ and $Q_s(M_r^S, x^*) = 1$, output 1. Otherwise, output 0.*

Definition 14. *Let an (n, t, ϵ)-adversarial* ***reveal*** *resilient steady Learned Bloom Filter be any steady Learned Bloom Filter for which it holds that, $\forall \lambda > n \in \mathbb{N}, P[\Lambda^{2l}_{A,t}(\lambda) = 1] \leq \epsilon$.*

5 Security

We attempt two solutions to the problem of securing the Learned Bloom Filter. Our first solution combines a Learned Bloom Filter technique called "sandwiching" introduced by [8] and the use of a pseudo-random permutation on the Bloom Filter input set first proposed by [3]. Our second solution relies on the (non Sandwiched) Learned Bloom Filter and instead employs the use of pseudo-random permutations to two Backup Bloom Filters that we introduce as part of the construction.

In Section 5.1, we first provide a review of the setting from Section 4.2. Then we formally define a Sandwiched Learned Bloom Filter. Finally we provide a brief introduction to pseudo-random permutations. We introduce the Uptown Bodega Filter in Section 5.2. We then discuss concerns regarding the utility of hte Uptown Bodega Filter which lead us to the Downtown Bodega Filter, which we introduce in Section 5.3.

5.1 Preliminaries

Setting Review: Consider a set of elements $\mathcal{K} \subset S$ and a set of elements $\mathcal{U}$ such that $\forall u \in \mathcal{U}, u \notin S$. We form a dataset $\mathcal{D} = \{(x_i, y_i = 1) | x_i \in \mathcal{K}\} \cup \{(x_i, y_i = 0) | x_i \in \mathcal{U}\}$. Let L_r be any construction algorithm for l. We train a Bloom Filter Learning Model, l, by running algorithm L_r on $\mathcal{D}$. Let $l(x)$ be the probability estimate from the learning model (Definition 10) that x is an element in S. A value τ may be chosen as a threshold. When $l(x) \geq \tau$ then the Learned Bloom Filter considers x to be an element of S. Otherwise, the Learned Bloom Filter passes x on to the classical Backup Bloom Filter.

Definition 15. *Let a Sandwiched Learned Bloom Filter, $SB_r = (IC_r, IQ, l, \tau, C_r, Q)$ where l obeys Definition 10, τ is a given threshold for l, IC_r and C_r obey Definition 2, and IQ and Q obey either Definition 3 or Definition 4. (IC_r, IQ) form a classical Bloom Filter (Definition 5), which we referr to as an Initial Bloom Filter. The Initial Bloom Filter holds the set S. (C_r, Q) also form a classical Bloom Filter, which we refer to as a Backup Bloom Filter. The Backup Bloom Filter holds the set $\{x : x \in S | l(x) < \tau\}$.*

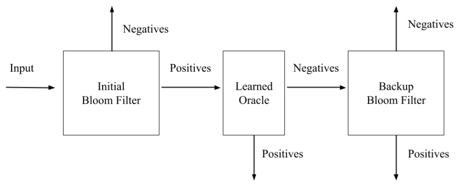

Fig. 3: A Sandwiched Learned Bloom Filter. The initial filter only allows positives (true positive and false positive) to reach the Learned Bloom Filter.

Figure 3 shows an example of a Sandwiched Learned Bloom Filter.

Lemma 1. *If x is a false positive in a Sandwiched Learned Bloom Filter, SB_r, then x is a false positive in the Initial Bloom Filter (IC_r, IQ).*

Proof. The proof follows from the definition of a Sandwiched Learned Bloom Filter.

We now introduce pseudorandom permutations, adapted from Chapter 3 of [13].

Definition 16. *Let an efficient permutation F be any permutation for which there exists a polynomial time algorithm to compute $F_k(x)$ given k and x, and there also exists a polynomial time algorithm to compute $F_k^{-1}(x)$ given k and x.*

Definition 17. *Let $F : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^*$ be an efficient, length-preserving, keyed function. F is a keyed permutation if $\forall k$, $F_k(\cdot)$ is one-to-one.*

Definition 18. *Let $F : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^*$ be an efficient keyed permutation. F is a pseudorandom permutation if for all probabilistic polynomial time distinguishers D, there exists a negiligible function* $\texttt{negl}$*, such that* $|\Pr[D^{F_k(\cdot)F_k^{-1}(\cdot)}(1^n) = 1] - \Pr[D^{f_n(\cdot)f_n^{-1}(\cdot)}(1^n) = 1]| \leq \texttt{negl}(n)$

Theorem 4.8 of [3] proves that for a classical steady (n, ϵ)-Bloom Filter that uses m bits of memory, if pseudorandom permutations exist, then there exists a negiligible function $\texttt{negl}$ such that for security parameter λ there exists a $(n, \epsilon + \texttt{negl}(\lambda))$-adversarial resilient Bloom Filter that uses $m^{'} = m + \lambda$ bits of memory. This secure Bloom Filter can be constructed by running the initialization algorithm on $S^{'} = \{F_k(x) : x \in S\}$ instead of S [3].

5.2 The Uptown Bodega Filter

Now, we introduce a secure Learned Bloom Filter built on top of a Sandwiched Learned Bloom Filter and a pseudorandom permutation which we call a Uptown Bodega Filter

Definition 19. *Let a Uptown Bodega Filter be a data structure $BB_{r,k} = (F_k, IC_r, IQ, l, \tau, C_r, Q)$ such that F_k is a pseudorandom permutation and $(IC_r, IQ, l, \tau, C_r, Q)$ is a Sandwiched Bloom Filter representing the set $S^{'} = \{F_k(x) : x \in S\}$ in the steady setting.*

Theorem 1. *Let SB_r be an (n, ϵ)-Sandwiched Bloom Filter using m bits of memory. If pseudorandom permutations exist, then there exists a negligible function* $\text{negl}(\cdot)$ *such that for security parameter λ there exists an $(n, \epsilon + \text{negl}(\lambda))$-adversarial resilient Uptown Bodega filter that uses $m^{'} = m + \lambda$ bits of memory.*

Our proof follows the same line of reasoning as Theorem 4.8 of [3] which proves that, if pseudorandom permutations exist, any probabilistic polynomial time adversary cannot distinguish between the classical Bloom Filter with pseudorandom permutations and Challenge 1 by more than a negligible advantage. Due to Lemma 1, this result also holds for the Uptown Bodega Filter. The crux of the proof relies on the fact that running F_k on an adversary's queries permits us to consider the queries as random and not chosen adaptively by an adversary, while having no effect on the correctness of the Uptown Bodega Filter.

Proof. We construct a Uptown Bodega Filter $UB_{k,r}$ from a Sandwiched Bloom Filter SB_r as follows. Choose the first key $k \in \{0,1\}^{\lambda}$ from pseudorandom permutation, F_k, over $\{0,1\}^{\log|U|}$. Let $S_k = F_k(S) = \{F_k(x) : x \in S\}$. We initialize SB_R with S_k. For each input x, our query algorithm outputs $SB_{k,r}(F_k(x))$. Just like in the classical Bloom Filter construction of [3], the only additional memory required is to store the key k for F_k, which is λ bits. Similarly, the running time of the query algorithm of the Uptown Bodega Filter is one pseudo-random permutation more than the running time of the query algorithm of the Sandwiched Bloom Filter.

The completeness of $UB_{k,r}$ follows from the completeness of $SB_{k,r}$. If $x \in S$ then $SB_{k,r}$ was initialized with $F_k(x)$ which will return 1 from the completeness of SB_r. The resilience of the construction follows from the following argument: Consider an experiment where F_K in the Uptown Bodega Filter is replaced by a truly random permutation oracle $R(\cdot)$. Since x has not been queried, we know that $R(x)$ is a truly random element that was not queried before, and we may think of it as chosen prior to the initialization of $SB_{k,r}$. From the soundness of SB_r we get that the probability of x being a false positive is at most ϵ.

Now we show that no probabilistic polynomial time (PPT) adversary A can distinguish between the Uptown Bodega Filter we constructed using $R(\cdot)$ and the Uptown Bodega Filter construction that uses the pseudorandom permutation F_k by more than a negligible advantage. Suppose that there does exist a polynomial $p(\lambda)$ such that A can attack $UB_{k,r}$ and find a false positive with probability $\epsilon + \frac{1}{p(\lambda)}$. We can run A on $UB_{k,r}$ where the oracle is replaced by an oracle that is either random or pseudorandom. We return 1 if A successfully finds a false positive. This implies that we may distinguish between a truly random permutation and a pseudorandom permutation with probability $\geq \frac{1}{p(\lambda)}$. This contradicts the indistinguishability of pseudorandom permutations.

Utility of the Uptown Bodega Filter While we have shown the existence of (n, ϵ)-adversarial resilient Uptown Bodega Filters, it is unclear whether there is any utility in using an Uptown Bodega Filter over a secure classical Bloom Filter. This is due to the fact that we train our learning model on $F_k(x) \in S$, a pseudorandom permutation of the set S, instead of training it directly on S. Intuitively, any structure in S that can be "learned" by a Bloom Filter Learning Model is destroyed by taking the psuedorandom permutation of S. In other words, if pseudorandom permutations exist, then to any learning model that runs in Probabilistic Polynomial Time (PPT), $F_k(x) \in S$ should appear the same as $|S|$ values sampled from a truly random oracle $R(\cdot)$. A precise proof for this is beyond the scope of this paper.

5.3 The Downtown Bodega Filter

To address the utility concerns of the Uptown Bodega Filter, we introduce a second secure Learned Bloom Filter which we call the Downtown Bodega Filter.

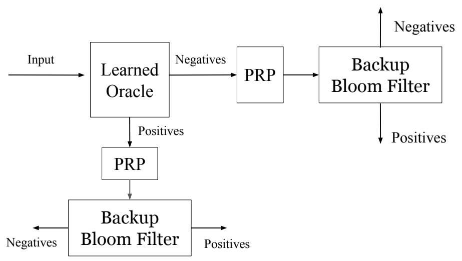

Fig. 4: A Downtown Bodega Filter. Both Positives and Negatives results for the Learned Oracle are routed Backup Bloom Filters secured with Pseudorandom Permutations

Definition 20. *Let a Downtown Bodega Filter be a data structure* $DB_{r,k_A,k_B} = (l, \tau, F_{k_A}, AC_r, AQ, F_{k_B}, BC_r, BQ)$ *such that 1)* F_{k_A} *and* F_{k_B} *are pseudorandom permutations, 2)* l *is a Bloom Filter Learning Model representing the set* S *with threshold* τ*, 3)* (AC_r, AQ) *and* (BC_r, BQ) *are classical Bloom Filters representing the sets* $AS^{'} = \{F_{k_A}(x) : x \in S\}$ *and* $BS^{'} = \{F_{k_B}(x) : x \in S\}$ *respectively in the steady setting.*

For convenience, we will refer to constructions (F_{k_A}, AC_r, AQ) and (F_{k_B}, BC_r, BQ) as Backup Bloom Filter A and Backup Bloom Filter B respectively.

Lemma 2. *Let B_r be an (n, ϵ)-Bloom Filter using m bits of memory. If pseudorandom permutations exist, then there exists a negligible function* $\operatorname{negl}(\cdot)$ *such that for security parameter λ, constructions (F_{k_A}, AC_r, AQ) and (F_{k2}, BC_r, BQ) are $(n, \epsilon + \operatorname{negl}(\lambda))$-adversarial reveal resilient Bloom Filters each using $m^{'} = m + \lambda$ bits of memory.*

Lemma 2 is just a rephrasing of Theorem 4.8 of Naor and Yogev [3] and follows directly from the theorem. We include a self-contained proof here for completeness. We also make it more evident that the proof holds not just for adversarial resilient Bloom Filters but also for adversarial **reveal** resilient Bloom Filter that provide guarantees under a strictly stronger adversarial model.

Proof. Let us consider the first construction Backup Bloom Filter A, (F_{k_A}, AC_r, AQ) (the second construction can be proven in the same way). We first choose a key $k_A \in \{0,1\}^\lambda$ for the pseudorandom permutation F_{k_A} over $\{0,1\}^{log|U|}$. Let $AS^{'} = \{F_{k_A}(x) : x \in S\}$. Our construction algorithm AC_r merely intializes B with $AS^{'}$. Our query algorithm on input x queries for $x^{'} = F_{k_A}(x)$. The only additional memory required is for storing k_A which is λ bits long.

The completeness follows from the completeness of B_r. The resilience of the construction follows from the following argument: consider an experiment where F_{k_A} in Backup Bloom Filter A is replaced by a truly random oracle $\mathcal{R}(\cdot)$. Since x haas not been queries, we know that $R(x)$ is a truly random element that was not queried before, and we may think of it as chosen prior to the initialization of Backup Bloom Filter A. From the soundness of Backup Bloom Filter A, we get that the probability of x being a false positive is at most ϵ.

Now we show that no probabilistic polynomial time (PPT) adversary A can distinguish between the Backup Bloom Filter A we constructed using $\mathcal{R}(\cdot)$ and the Backup Bloom Filter A construction that uses the pseudorandom permutation F_{k_A} by more than a negiligible advantage. Suppose that there does exist a polynomial $p(\lambda)$ such that A can attack Backup Bloom Filter A and find a false positive with probability $\epsilon + \frac{1}{p(\lambda)}$. We can run A on Backup Bloom Filter A where the oracle is replaced by an oracle that is either random or pseudorandom. We return 1 if A successfully finds a false positive. This implies that we may distinguish between a truly random permutation and a pseudorandom permutation with probability $\geq \frac{1}{p(\lambda)}$. This contradicts the indistinguishability of pseudorandom permutations.

Theorem 2. *Let B_r be an (n, ϵ)-Sandwiched Bloom Filter using m bits of memory. If pseudorandom permutations exist, then there exists a negligible function* $\operatorname{negl}(\cdot)$ *such that for security parameter λ there exists an $(n, \epsilon + \operatorname{negl}(\lambda))$-adversarial reveal resilient Downtown Bodega filter that uses $m^{'} = m + 2\lambda$ bits of memory.*

Proof. We construct a Downtown Bodega Filter $DB_{k_A,k_B,r}$ from a Sandwiched Bloom Filter SB_r as follows. We use the memory budget of the initial classical Bloom Filter (IC_r, IQ) to construct (AC_r, AQ). We use the memory budget of the backup classical Bloom Filter (C_r, Q) to construct (BC_r, BQ). We do not modify the Bloom Filter Learning Model l which remains trained on set S with threshold τ. We choose keys $k_A, k_B \in \{0,1\}^\lambda$ and use 2λ bits of extra memory to store them.

The completeness of $DB_{k,r}$ follows from 1) the completeness of Backup Bloom Filter B and 2) the fact that any x such that $l(x) \leq \tau$ is declared not in S by the Downtown Bodega Filter if and only if $BQ(M, F_{k_B}(x))$ is also 0.

To prove the resilient of the construction, we first show that the security of the Downtown Bodega Filter construction is reducible to the security of Backup Bloom Filter A and Backup Bloom

Filter B. Consider a false positive i.e an $x \notin S$ for which the Downtown Bodega Filter returns 1. From the definition of the Downtown Bodega Filter, one of the following two cases must be true

- **Case 1**: The Learned Oracle returned 1 and Backup Bloom Filter A returned 1, more precisely, $l(x) \geq \tau \wedge AQ(F_{k_A}(x)) = 1$
- **Case 2**: The Learned Oracle returned 0 and Backup Bloom Filter B returned 1, more precisely, $l(x) < \tau \wedge BQ(F_{k_B}(x)) = 1$

Therefore, for any probabilistic polynomial time adversary to induce a false positive in the overall Downtown Bodega Filter construction, they must either induce a false positive in Backup Bloom Filter A or Backup Bloom Filter B. We have already proven in Lemma 2 that both Backup Bloom Filter A and Backup Bloom Filter B are $(n, \epsilon + \text{negl}(\lambda))$-adversarial reveal resilient. It follows that the entire construction is $(n, \epsilon + \text{negl}(\lambda))$-adversarial reveal resilient. This concludes our proof.

6 Discussion

In this section, we discuss the only two known attacks on the Learned Bloom Filter, introduced by Reviriego et al [2]. We refer to Attack 1 from their work as the Blackbox Mutation Attack, and Attack 2 from their work as the Whitebox Mutation Attack respectively. We discuss the Blackbox Mutation Attack in Section 6.1, and the Whitebox Mutation Attack in Section 6.2. Both sections include details on how our Bodega Filter constructions mitigate the attack.

6.1 Blackbox Mutation Attack

The blackbox adversarial model defined by Reviriego et al [2] is slightly weaker but very similar to the adversarial model we define in Challenge 3 of Section 4.2. Both our adversarial model and Reviriego et al's blackbox adversary model allows the adversary access to query the Learned Bloom Filter. One major difference is that our model allows the adversary to choose the initial set S that is represented by the Learned Bloom Filter, whereas Reviriego et al's model does not. In their attack, Reviriego et al first test elements until a positive (whether a false positive or true positive) is found. They then *mutate* the positive by changing a small fraction of the bits in the input in order to generate more false positives. The attack targets the Learned Oracle (recall Figure 2) by making it generate false positives without the input reaching the Backup Bloom Filter.

Since, as we mentioned, Reviriego et al's attack relies on the fact that the adversary can query the Learned Oracle and make it generate false positives without any involvement of the Backup Bloom Filter, a Sandwiched Bloom Filter construction is intuitively more resilient against this kind of mutation attack. Since the Uptown Bodega Filter takes a pseudorandom permutation of the input x prior to sending it to either the Learned Oracle or the Bloom Filters, the mutation approach is no longer viable. Unlike in the Uptown Bodega Filter, the Downtown Bodega Filter *does* give the adversary the ability to query the Learned Oracle directly. However, the extra (pseudo-random permuted) Backup Bloom Filter for the case $l(x) > \tau$ lowers the probability of mutation-based false positives from the Learned Oracle ending up as false positives in the overall construction.

6.2 Whitebox Mutation Attack

The whitebox adversarial model defined by Reviriego et al [2] is similar to the adversarial model we define in Challenge 4 in Section 4.2. With knowledge of the state of the Learned Oracle, the adversary can generate mutations in a more sophisticated way. Reviriego et al provide the example of a malicious URL dataset where an adversary may begin with a non-malicious URL and make changes such as removing the “s” in “https” or removing the “www” to generate false positives. Since we have shown the Downtown Bodega Filter to be (n, t, ϵ)-adversarial reveal resilient in the steady setting, such mutations will not provide the adversary any advantage over the construction.

7 Hybrid Adversarial Model

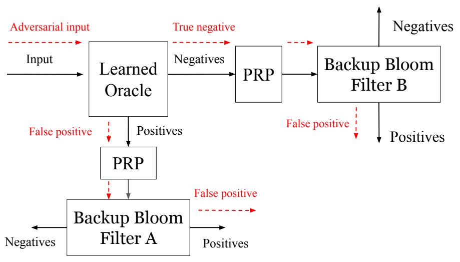

Fig. 5: To generate a false positive in the Downtown Bodega Filter, the adversary must either 1) generate a false positive in the Learned Oracle and direct their query through Backup Bloom Filter A or 2) generate a true negative in the Learned Oracle and direct their query through Backup Bloom Filter B

Consider a streaming workload sent to a Bloom Filter consisting of N queries. An adversary chooses exactly αN of those queries, where $\alpha \in [0, 1]$. For an adversarial query to generate a false positive in the Downtown Bodega Filter, one of the following must hold true for a given query (see Figure 5):

1. The query generates a false positive in the Learned Oracle and a false positive in Backup Bloom Filter A
2. The query generates a true negative in the Learned Oracle and a false positive in Backup Bloom Filter B

We first model the false positive probability of the Downtown Bodega Filter as a function of its memory budget.

7.1 Downtown Bodega Filter

Without loss of generality, let α_P of the adversarial queries be such that they generate false positives in the Learned Oracle and go through Backup Bloom Filter A. Similarly, let α_N of the adversarial queries be such that they generate true negatives and go through Backup Bloom Filter B. Note that $\alpha = \alpha_P + \alpha_N$.

Let $\mathrm{FPR}(m)$ be the expected false positive probability of a classical Bloom Filter with memory budget m. Let $\mathrm{FPR}_L(m)$ be the expected false positive probability of a Learned Oracle L with memory budget m. Similarly, let $\mathrm{TNR}_L(m)$ be the expected true negative probability of a Learned

Oracle L with memory budget m. We make the assumption that the correctness probability of the Learned Oracle is independent of the correctness probability of the Backup Bloom Filters. In particular, we assume that for any m, m', $\text{FPR}_L(m) \cap \text{FPR}(m') = \text{FPR}_L(m)\text{FPR}(m')$ and $\text{TNR}_L(m) \cap \text{FPR}(m') = \text{TNR}_L(m)\text{FPR}(m')$.

Consider a system where the total memory budget is M. Let the memory allocation of a Downtown Bodega Filter from memory budget M be assigned as follows. Let m_L be the number of bits of memory assigned to a Learned Oracle L. Let m_A be the number of bits of memory assigned to Backup Bloom Filter A. Let m_B be the number of bits of memory assigned to Backup Bloom Filter B. Let λ be the number of bits assigned to the key of the pseudorandom permutations used prior to Backup Bloom Filter A and Backup Bloom Filter B. Note that to stay within the memory budget it must hold that $M \geq m_L + m_A + m_B + 2\lambda$.

Theorem 3. *The Downtown Bodega Filter provides an expected false positive probability of*

$$FPR_{DB}(M) = FPR_L(m_L)FPR(m_A) + TNR_L(m_L)FPR(m_B)$$

where the probability is taken over the random coins of the pseudorandom permutations, the random coins used in the construction of the Learned Oracle, the random coins used in the construction of Backup Bloom Filters A and B, and the random coins used in the generation of the non-adversarial queries.

Proof. From the definition of a Downtown Bodega Filter it follows that a false positive in the overall construction must either be a false positive in Backup Bloom Filter A or a false positive in Backup Bloom Filter B. If the query is a false positive in Backup Bloom Filter A, it must also be a false positive in the Learned Oracle. Alternatively, if the query is a false positive in Backup Bloom Filter B, it must be a true negative in the Learned Oracle. The result follows.

We now derive an expression for the false positive probability of the Downtown Bodega Filter in the hybrid adversarial setting. Recall that we are assuming that out of N queries, the adversary makes α_P queries that generate a false positive in the Learned Oracle and α_N queries that generate a true negative in the Learned Oracle.

Theorem 4. *In the hybrid adversarial setting, the expected false positive probability of the Downtown Bodega Filter is*

$$\alpha_P FPR(m_A) + \alpha_N FPR(m_B) + (1 - \alpha_P - \alpha_N)FPR_{DB}(M)$$

where the probability is taken over the random coins of the pseudorandom permutations, the random coins used in the construction of the Learned Oracle, the random coins used in the construction of Backup Bloom Filters A and B, and the random coins used in the generation of the non-adversarial queries.

Proof. For each query i among N queries, one of the following cases holds.

Case 1: The query is not an adversary generated query. Therefore as established by Theorem 3, the False Positive Probability for the query is $\text{FPR}_{DB}(M)$. There are $(1 - \alpha_P - \alpha_N)N$ such queries.

Case 2: The query is an adversary generated query such that it generates a false positive in the Learning Oracle. Since the Learning Oracle generating a false positive and the Learning Oracle generating a true negative are mutually exclusive events, the False Positive Probability for the query is the False Positive Probability of Backup Bloom Filter A i.e $\mathrm{FPR}(m_A)$. There are $\alpha_P N$ such queries.

Case 3: The query is an adversary generated query such that it generates a true negative in the Learning Oracle. Following logic similar to case 2, we can derive the False Positive Probability of the query to be $\mathrm{FPR}(m_B)$. There are $\alpha_N N$ such queries.

The expected false positive probability of N queries is then $\frac{1}{N}(\alpha_P N \cdot \mathrm{FPR}(m_A) + \alpha_N N \cdot \mathrm{FPR}(m_B) + (1 - \alpha_P - \alpha_N)N \cdot \mathrm{FPR}_{DB}(M))$. The statement of the theorem follows.

7.2 Secure Classical Bloom Filter

An alternative construction is to simply use a well-tuned (n, t, ϵ)-adversarial reveal resilient classical Bloom Filter. We will refer to this construction in this section as the Secure Classical Bloom Filter without causing confusion. The expected false positive probability of the Secure Classical Bloom Filter is then $\mathrm{FPR}(m_L + m_A + m_B + \lambda)$. The extra λ is due to the fact that the Secure Classical Bloom Filter requires one less pseudorandom permutation than the Downtown Bodega Filter. The expected false positive probability of the Secure Classical Bloom Filter is, by definition, $\mathrm{FPR}(m_L + m_A + m_B + \lambda)$.

We now provide an expression that encapsulates all the cases in the hybrid adversarial setting where a Downtown Bodega Filter construction provides a lower false positive probability compared to a secure Classical Bloom Filter construction for the same memory budget.

Theorem 5. *For a given memory budget $M = m_L + m_A + m_B + 2\lambda$, in the hybrid adversarial setting with given $\alpha = \alpha_P + \alpha_N$, the expected false positive probability of the Downtown Bodega Filter is lower than the expected false positive probability of the Secure Classical Bloom Filter if the following holds true:*

$$\alpha_P FPR(m_A) + \alpha_N FPR(m_B) + (1 - \alpha_P - \alpha_N) FPR_{DB}(M) < FPR(m_L + m_A + m_B + \lambda)$$

where the probability is taken over the random coins of the pseudorandom permutations, the random coins used in the construction of the Learned Oracle, the random coins used in the construction of Backup Bloom Filters A and B, the random coins used in the construction of the Secure Classical Bloom Filter, and the random coins used in the generation of the non-adversarial queries.

Proof. The proof follows directly from the expression derived for the expected false positive probability of the Downtown Bodega Filter construction in Theorem 4 and the expression for the expected false positive probability of the Secure Classical Bloom Filter.

7.3 Tradeoffs

From [7], we know that the false positive probability for a Bloom Filter with m bits encoding a set S of cardinality n, using k hash functions is

$$\mathrm{FPR}(m) = (1 - e^{-kn/m})^k$$

In our analysis, we use the value of the number of hash functions k is always optimally chosen to be $\ln 2 \cdot (m/n)$ [7] and n is constant. We further treat a Learned Oracle as a Bloom Filter with better false positive probability for the same memory budget, consistent with assumptions made by prior work [1] [8].

$$\text{FPR}_L(m) = c(1 - e^{-kn/m})^k$$

where $c \leq 1$.

We note that the true negative probability of a Learned Oracle is merely the probability of a negative entry (which is constant as we are assuming set S is constant) is not marked as a false positive. Let Q_N be the fraction of true negative non-adversarial queries. We have $\text{TNR}_L(m) = (1 - \text{FPR}_L(m))Q_N = (1 - c(1 - e^{-kn/m})^k)Q_N$.

To evaluate how much lower the false positive probability of a Learned Oracle needs to be for the Downtown Bodega Filter to perform better than the Secure Classical Bloom Filter, we may then use these derivations in Theorem 5.

7.4 Realistic Example

We choose realistic values for our example from prior work on evaluating Learned Bloom Filters [1] on Google's transparency report. We pick 2 Megabytes as our memory budget, m, chosen from the range of values in Figure 10 of [1]. We choose the cardinality of the set we want to encode, n, as 1.7 million based on the number of unique URLs in Google's transparency report evaluated in [1]. With a memory budget of 2 Megabytes, [1] demonstrate that a Learned Bloom Filter has 0.25 of the False Positive Ratio of a Classical Bloom Filter, hence we use that as our value for c (See Section 7.3).

Parameter	Explanation	Value
M	Memory Budget	2 Megabytes
m_L	Memory budget for Learned Oracle	1 Megabytes
m_A	Memory budget for Backup Bloom Filter A	0.5 Megabytes
m_B	Memory budget for Backup Bloom Filter B	0.5 Megabytes
n	Cardinality of set to encode	1.7 Million
c	FPR of Learned Bloom Filter / FPR of Classical Bloom Filter	0.25
λ	Number of bits in secret key	128 bits

Table 1: A summary of the chosen values for our realistic example of the performance tradeoffs of a Downtown Bodega Filter compared to a Secure Classical Filter in the hybrid adversarial case

We use 128 bits as the size of our security parameter, λ. For the case of the Downtown Bodega Filter, we let the Learned Oracle take 1 Megabytes, while dividing the remaining 1 Megabytes equally between Backup Bloom Filters A and B. Lastly, we take α to be a variable ranging from 0 to 0.2 equally divided between α_P and α_N. Our chosen values are summarized in Table 1. Figure 6 shows the results of our calculations. As can be seen, when the adversary has access to less than a certain cutoff fraction of the workload, the Downtown Bodega Filter outperforms the Secure Classical Bloom Filter for the same memory budget.

Acknowledgments. The authors would like to thank Hussain Tirmazi for intitial feedback on the paper

Disclosure of Interests. The authors have no competing interests

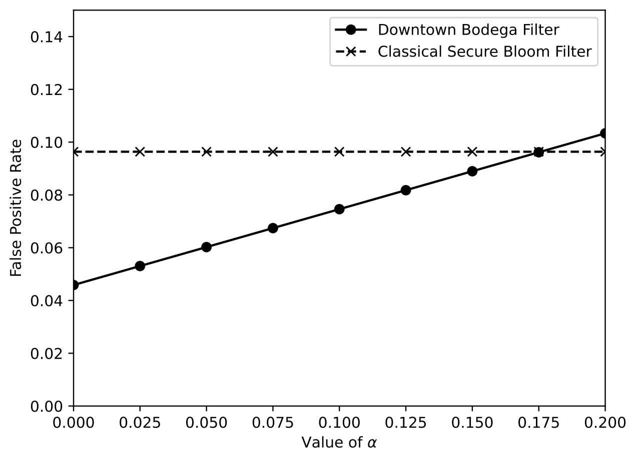

Fig. 6: The False Positive Rate of the Downtown Bodega Filter compared to the Secure Classical Bloom Filter in the hybrid adversarial setting where the adversary chooses between 0 and 0.2 of the workload.

Bibliography

[1] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, “The case for learned index structures,” in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 489–504. [Online]. Available: https://doi.org/10.1145/3183713.3196909

[2] P. Reviriego, J. Alberto Hernández, Z. Dai, and A. Shrivastava, “Learned bloom filters in adversarial environments: A malicious url detection use-case,” in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, 2021, pp. 1–6.

[3] M. Naor and Y. Eylon, “Bloom filters in adversarial environments,” *ACM Trans. Algorithms*, vol. 15, no. 3, jun 2019. [Online]. Available: https://doi.org/10.1145/3306193

[4] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Commun. ACM*, vol. 13, no. 7, p. 422–426, jul 1970. [Online]. Available: https://doi.org/10.1145/362686.362692

[5] 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.

[6] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang, “On the false-positive rate of bloom filters,” *Information processing letters*, vol. 108, no. 4, pp. 210–213, 2008.

[7] M. Mitzenmacher and A. Broder, “Network applications of bloom filters: A survey,” *Internet Mathematics Journal*, 2004.

[8] M. Mitzenmacher, “A model for learned bloom filters, and optimizing by sandwiching,” in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS’18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 462–471.

[9] 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.

[10] T. Gerbet, A. Kumar, and C. Lauradoux, “The power of evil choices in bloom filters,” in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 101–112.

[11] D. Clayton, C. Patton, and T. Shrimpton, “Probabilistic data structures in adversarial environments,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1317–1334. [Online]. Available: https://doi.org/10.1145/3319535.3354235

[12] M. Filic, K. G. Paterson, A. Unnikrishnan, and F. Virdia, “Adversarial correctness and privacy for probabilistic data structures,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1037–1050. [Online]. Available: https://doi.org/10.1145/3548606.3560621

[13] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*, 2nd ed. Chapman & Hall/CRC, 2014.